

Les fonctions SQL

Table des matières

I. Contexte	3
II. Fonctions numériques et fonctions textes	3
III. Exercice : Appliquez la notion	6
IV. Fonctions de date	7
V. Exercice : Appliquez la notion	9
VI. Fonctions de changement de type	10
VII. Exercice : Appliquez la notion	12
VIII. Fonctions d'agrégation	12
IX. Exercice : Appliquez la notion	14
X. Expressions conditionnelles	14
XI. Exercice : Appliquez la notion	16
XII. Essentiel	16
XIII. Auto-évaluation	17
A. Exercice final	17
B. Exercice : Défi	18
Solutions des exercices	19

I. Contexte

Durée : 1 h

Environnement de travail : MariaDB, DataGrip

Pré-requis : Connaître les requêtes simples

Contexte

Récupérer les données telles quelles dans une base de données est utile, mais rarement suffisant.

Les bases de données fournissent heureusement de nombreux opérateurs et fonctions afin de réaliser :

- les opérations usuelles : addition, soustraction...
- des opérations de texte
- des opérations de date
- des opérations de décompte, de somme, de moyenne
- des traitements différents selon certaines conditions

II. Fonctions numériques et fonctions textes

Objectifs

- Effectuer des opérations arithmétiques simples (addition, soustraction...)
- Effectuer des opérations arithmétiques avancées (racine carrée, valeur absolue...)
- Arrondir des valeurs numériques
- Tronquer et concaténer du texte

Mise en situation

La plupart des données qui vont être stockées dans notre base vont être sous forme de nombres ou de chaînes de caractères. Le langage SQL dispose de nombreuses fonctions permettant de réaliser toutes sortes d'opérations sur ces types de données, allant de simples opérations mathématiques à des manipulations plus poussées.

Opérateurs numériques

Voici quelques opérateurs parmi les plus fréquents pour effectuer des opérations sur des nombres :

- `+` pour l'addition
- `-` pour la soustraction
- `*` pour la multiplication
- `/` pour la division
- `DIV` pour la division entière, donc la partie entière d'une division
- `MOD` pour le modulo, donc le reste d'une division

Exemple

```
1 SELECT 1 + 2; -- Retourne 3
2 SELECT 4 - 3; -- Retourne 1
3 SELECT 3 * 2; -- Retourne 6
4 SELECT 7 / 2; -- Retourne 3.5
5 SELECT 7 DIV 2; -- Retourne 3
6 SELECT 7 MOD 2; -- Retourne 1
```

Autres exemples plus concrets :

```
1 SELECT colonne_prix + colonne_TVA FROM nom_table ; -- Retourne l'addition des deux
2 SELECT colonne_prix - colonne_reduction FROM nom_table ; -- Retourne la soustraction des deux
3 SELECT colonne_prix * colonne_quantité FROM nom_table ; -- Retourne le produit des deux
```

Rappel Les fonctions

Une fonction est un objet qui renvoie une valeur typée. Elle peut recevoir un ou plusieurs paramètres.

En SQL, une fonction est appelée par son nom, suivi de parenthèses. D'éventuels paramètres peuvent être insérés entre ces parenthèses, et doivent être séparés par des virgules.

Fonctions numériques

Voici quelques fonctions numériques parmi les plus courantes. Toutes ces fonctions ne prennent qu'un seul paramètre, à savoir le nombre sur lequel faire l'opération.

- ABS : valeur absolue d'un nombre.
- ROUND, FLOOR, CEIL : arrondit le nombre respectivement au plus proche, au-dessous et au-dessus.
- SIGN : retourne le signe d'un nombre, à savoir 1 pour un nombre positif, 0 pour un nombre nul, ou -1 pour un nombre négatif.
- SQRT : racine carrée d'un nombre.

Exemple

```
1 SELECT ABS(-3); -- Retourne 3
2 SELECT ROUND(3.4), ROUND(3.6); -- Retourne 3 et 4
3 SELECT FLOOR(3.4), FLOOR(3.6); -- Retourne 3 et 3
4 SELECT CEIL(3.4), CEIL(3.6); -- Retourne 4 et 4
5 SELECT SIGN(3.4), SIGN(-3.4), SIGN(0); -- Retourne 1, -1 et 0
6 SELECT SQRT(4), SQRT(2); -- Retourne 2 et 1.414213562...
```

Concaténer du texte

Pour concaténer du texte, c'est-à-dire assembler deux chaînes de caractères, il faut employer la fonction CONCAT ou CONCAT_WS :

- CONCAT concatène simplement les textes passés en paramètres.
- CONCAT_WS rajoute un séparateur entre les textes, WS signifiant « *with separator* ». Le premier paramètre passé est le séparateur qui sera rajouté entre tous les textes suivants.

```
1 SELECT CONCAT('Un texte', 'à concaténer'); -- Retourne "Un texteà concaténer"
2 SELECT CONCAT_WS(' ', 'Un texte', 'à concaténer'); -- Retourne "Un texte à concaténer"
```

Attention

De nombreux langages de programmation permettent d'utiliser l'opérateur + comme opérateur de concaténation de chaîne. Cela ne fonctionne pas en SQL.

De plus, la façon pour concaténer dépend du SGBD utilisé. La syntaxe donnée ici fonctionne dans MariaDB, et dans certains autres SGBD comme PostgreSQL. Il faut se référer à la documentation du SGBD utilisé.

Découper du texte

Il existe plusieurs fonctions pour récupérer une sous-chaîne :

- `LEFT` et `RIGHT` prennent en paramètres un texte et une longueur et permettent de récupérer respectivement les parties gauche et droite du texte de la longueur donnée.
- `MID`, `SUBSTR`, `SUBSTRING` prennent en paramètres un texte, une position de début et une longueur et permettent de récupérer une sous-chaîne à partir de la position de début et de la longueur donnée (ces trois fonctions sont équivalentes).
- `SUBSTRING_INDEX` prend en paramètres un texte, un délimiteur et un nombre, et permet de récupérer la partie gauche d'un texte jusqu'à ce qu'on ait trouvé un certain nombre de fois le délimiteur, qui est exclu du résultat final. Par exemple, on peut récupérer un certain nombre de phrases « jusqu'au point ».

```
1 SELECT LEFT('Un long texte', 5), RIGHT('Un long texte', 5); -- Retourne "Un lo" et "texte"
2 SELECT MID('Un long texte', 4, 6); -- Retourne "long t"
3 SELECT SUBSTRING_INDEX('Un long texte', ' ', 1), SUBSTRING_INDEX('Un long texte', ' ', 2); --
   Retourne "Un" et "Un long"
```

Attention

Il ne faut pas oublier que la position du premier caractère d'une chaîne est 1 et pas 0, contrairement à ce qu'on retrouve dans les langages de programmation plus classiques.

Chercher dans du texte

- `LENGTH` permet de connaître la longueur d'une chaîne passée en paramètre.
- `LOCATE` prend en paramètres une sous-chaîne et une chaîne de caractères, et renvoie la position de la sous-chaîne dans la chaîne (0 si non trouvée).
- `LIKE` permet de comparer la chaîne à un modèle. Attention, `LIKE` n'est pas une fonction, mais un opérateur. Retourne 1 si la chaîne correspond au modèle, 0 sinon.

```
1 SELECT LENGTH('Un long texte'); -- Retourne 13
2 SELECT LOCATE('texte', 'Un long texte'); -- Retourne 9
3 SELECT 'Un long texte' LIKE '%texte%'; -- Retourne 1
```

Le caractère % avec l'opérateur `LIKE` signifie « toute valeur possible, y compris vide » :

- `LIKE '%texte%'` signifie « contient "texte" »
- `LIKE '%texte'` signifie « finit par "texte" »
- `LIKE 'texte'` signifie « est égal à "texte" »

Autres exemples plus concrets :

```
1 SELECT * FROM nom_table WHERE email LIKE '%.com'; -- Retourne tous les emails qui finiront
   par .com.
2 SELECT * FROM nom_table WHERE prenom LIKE 'A%'; -- Retourne tous les prénoms qui commencent
   par A
```

Modifier du texte

- UPPER transforme les caractères du texte passé en paramètre en majuscules.
- LOWER transforme les caractères du texte passé en paramètre en minuscules.
- TRIM retire les espaces inutiles au début et à la fin du texte passé en paramètre. Cette fonction ne retire pas les espaces séparant des mots.
- REPLACE prend en paramètres un texte, un caractère à rechercher et un caractère de remplacement, et permet remplacer toutes les occurrences du caractère à rechercher par l'autre.

```
1 SELECT UPPER('Du texte en minuscule'); -- Retourne "DU TEXTE EN MINUSCULE"
2 SELECT LOWER('Du texte en MAJUSCULE'); -- Retourne "du texte en majuscule"
3 SELECT TRIM(' 4 espaces au début et 3 à la fin '); -- Retourne "4 espaces au début et 3 à la fin"
4 SELECT REPLACE('ici et là', 'm', 'l'); -- Retourne "ici et là"
```

Fondamental

Même s'ils sont ici utilisés avec des valeurs simplistes, les opérateurs et les fonctions présentées dans ce cours permettent de manipuler les valeurs des colonnes d'une requête. Ainsi, il est possible d'écrire `SELECT quantity * price FROM order` pour calculer le prix final d'une commande d'un produit, ou `SELECT UPPER(lastName) FROM users` pour récupérer le nom d'un utilisateur en majuscules.

Cette liste de fonctions n'est pas exhaustive, mais représente simplement les fonctions les plus souvent utilisées. N'hésitez pas à fouiller dans la documentation en cas de besoin.

Syntaxe À retenir

- Les opérateurs et les fonctions habituels des langages de programmation sont aussi disponibles en SQL.
- Il existe de très nombreuses fonctions permettant de manipuler des nombres et des chaînes de caractères, qui sont identifiées par un nom et peuvent posséder des paramètres, passés entre parenthèses et séparés par des virgules.

Complément

Opérateurs¹

Fonctions numériques²

Fonctions mathématiques³

Fonctions de texte⁴

III. Exercice : Appliquez la notion

Question 1

[solution n°1 p.21]

Une table Utilisateurs contient deux colonnes : une colonne Nom et une colonne Prénom.

La requête suivante est utilisée pour renvoyer la liste des noms et prénoms contenus dans cette table.

```
1 SELECT Nom, Prenom FROM Utilisateurs
```

Transformez cette requête pour renvoyer le prénom suivi d'un espace suivi du nom, par exemple "Jean Dupont".

1 <https://dev.mysql.com/doc/refman/8.0/en/non-typed-operators.html>

2 <https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html>

3 <https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html>

4 <https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>

Question 2

[solution n°2 p.21]

Un trigramme est un assemblage de trois lettres permettant d'identifier un employé. C'est un procédé souvent utilisé par les entreprises comme un moyen simple de mentionner un collaborateur dans les documents.

Il existe plusieurs règles possibles pour former un trigramme, mais une règle populaire consiste à prendre la première lettre du prénom, et la première et dernière lettres du nom de famille. Par exemple, le trigramme de John Doe est JDE.

En vous basant sur la requête précédente, écrivez une requête permettant de retourner les trigrammes des utilisateurs.

Indice :

Il faut une seule colonne contenant les informations.

IV. Fonctions de date

Objectifs

- Comprendre comment les dates sont gérées par le SGBD
- Manipuler des dates en SQL

Mise en situation

Un autre type, plus particulier, que nous allons être amenés à manipuler sont les dates. Bien qu'elles soient représentées par des chaînes de caractères, elles sont à considérer comme un type à part entière.

Stockage des dates

Les dates, comme toute autre forme de données, sont conservées sous forme binaire dans la base de données. Une date affichée par la base de données est donc une représentation de cette donnée binaire après conversion au travers d'un format. De même, lorsque la base de données interprète une requête écrite incluant une date sous forme de texte, elle va tenter d'interpréter cette date.

Exemple

Dans la requête `SELECT '2025-05-15'`, la donnée présente n'est pas une date, mais un texte qui peut représenter une date pour un humain.

Lorsque c'est nécessaire, une base de données tentera de convertir automatiquement un texte en date. Par exemple, lors de l'exécution d'une requête possédant un `WHERE` sur un champ de type date, la base de données va comprendre que l'on souhaite comparer deux dates entre elles.

```
1 SELECT
2   *
3 FROM
4   Ventes
5 WHERE
6   DateVente = '2025-05-15' -- Ici, la donnée est bien une date si la colonne DateVente est
   de type DATE
```

Le problème de ce genre d'expression est qu'aucun format de date n'est fourni. Ici, le format utilisé est le format ISO 8601, qui a l'avantage de n'avoir aucune ambiguïté.

Conseil

Pour écrire une date dans une requête, il est très fortement recommandé d'utiliser le format ISO 8601 : AAAA-MM-JJ hh:mm:ss.

Cela évite toute confusion entre les jours et les mois, comme dans les formats MM-JJ-AAAA ou JJ-MM-AAAA.

Fonctions de création de date

Il est possible de créer une donnée qui soit **typée en tant que date** pour la base de données :

- NOW() et SYSDATE() renvoient la date et l'heure du serveur. Les deux fonctions sont équivalentes.
- MAKEDATE prend en paramètres une année et un numéro de jour et renvoie la date correspondante.
- MAKETIME prend en paramètres des heures, minutes et secondes et retourne le temps correspondant.
- STR_TO_DATE convertit le texte passé en paramètre en date, selon le format passé en second paramètre.

Méthode

Les formats de date

Les formats de date doivent être renseignés selon les spécifications de la base de données utilisée. Les codes sont fréquemment en anglais. Voici les codes pour MariaDB (les majuscules et minuscules sont importantes) :

- %Y pour les quatre chiffres de l'année
- %m pour les deux chiffres du mois
- %d pour les deux chiffres du jour
- %H pour heure de 00 à 23
- %i pour les minutes
- %s pour les secondes

Ainsi, le format ISO s'écrit %Y-%m-%d %H:%i:%s, ce qui donne une date sous la forme 2030-05-25 19:35:12.

L'ensemble des formats de date se trouve dans la documentation d'un SGBD. Voici pour MariaDB : https://mariadb.com/kb/en/date_format/.

Exemple

```
1 SELECT NOW(); -- Retourne la date du jour, typée au format date
2 SELECT MAKEDATE(2030, 45); -- Retourne la date 2030-02-14
3 SELECT MAKETIME(16, 45, 10); -- Retourne l'heure 16:45:10
4 SELECT STR_TO_DATE('2030-5-15', '%Y-%m-%d'); -- Retourne la date 2030-05-15, typée au format date
```

Manipuler des dates

Il n'est jamais simple de manipuler des dates. Il faut prendre en compte un grand nombre de paramètres, comme le nombre de jours d'un mois, les années bissextiles, etc.

Fort heureusement, MariaDB fournit un grand nombre de fonctions permettant de faire ça facilement :

- ADDDATE ajoute à la date passée en première paramètre le nombre de jours donné en second paramètre (qui peut être négatif pour soustraire un nombre de jours).
- DATE_ADD ajoute à la date passée en première paramètre un intervalle de temps, donné en second paramètre. Un intervalle est composé du mot-clé INTERVAL, suivi d'un nombre et d'une unité de mesure (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND...).
- DATEDIFF renvoie le nombre de jours entre deux dates passées en paramètre. Attention, cette fonction renvoie un nombre, et non une date.


```

1 SELECT ADDDATE('2030-05-15', 5); -- Retourne 2030-05-20
2 SELECT DATE_ADD('2030-05-15', INTERVAL 5 DAY); -- Retourne 2030-05-20
3 SELECT DATE_ADD('2030-05-15', INTERVAL 5 WEEK); -- Retourne 2030-06-19
4 SELECT DATEDIFF('2030-05-15', '2030-04-28'); -- Retourne 17

```

Extraire des composants de date

La base de données fournit également des fonctions permettant d'extraire des parties d'une date. Par exemple :

- YEAR, MONTH et DAY renvoient respectivement l'année, le mois et le jour d'une date
- DAYOFYEAR renvoie le numéro du jour dans l'année, de 1 à 365 ou 366
- WEEKOFYEAR renvoie le numéro de la semaine dans l'année

```

1 SELECT YEAR('2030-05-15'); -- Retourne 2030
2 SELECT MONTH('2030-05-15'); -- Retourne 5
3 SELECT DAY('2030-05-15'); -- Retourne 15
4 SELECT DAYOFYEAR('2030-05-15'); -- Retourne 135
5 SELECT WEEKOFYEAR('2030-05-15'); -- Retourne 20

```

Conseil

Tout SGBD fournit ce genre de fonctions, même si elles ne portent pas toujours le même nom ou n'ont pas la même syntaxe. Il est préférable de les utiliser plutôt que de vouloir convertir les dates en texte et de faire des extractions sur le texte.

Syntaxe À retenir

Un SGBD fournit de nombreuses fonctions pour faciliter la manipulation des dates. Parmi celles-ci, on retrouve les fonctions suivantes :

- Création de date : fonctions NOW(), MAKEDATE(), MAKETIME() et STR_TO_DATE()
- Manipulation : fonctions ADDDATE(), DATE_ADD() et DATEDIFF()
- Extraction : fonctions YEAR(), MONTH(), DAY()...

Complément

ISO 8601¹

Fonctions de date²

V. Exercice : Appliquez la notion

Question

[solution n°3 p.21]

La fonction NOW() renvoie la date actuelle :

```
1 SELECT NOW();
```

Écrivez une requête renvoyant séparément le numéro du jour, du mois et de l'année.

Indice :

Les fonctions de date correspondantes sont DAY, MONTH et YEAR.

1 https://fr.wikipedia.org/wiki/ISO_8601

2 <https://mariadb.com/kb/en/date-time-functions/>

VI. Fonctions de changement de type

Objectifs

- Changer le type d'une donnée
- Changer le format d'une donnée

Mise en situation

Une donnée possède toujours une valeur et un type de données. Ces deux informations sont indispensables pour interpréter une donnée. Par exemple, dans une base de données, le nombre 15 est à différencier d'une chaîne de caractères comportant les deux caractères 1 et 5.

Un SGBD peut effectuer certaines conversions par lui-même (comme pour l'interprétation des dates), mais il existe des fonctions permettant de forcer un changement de type.

Syntaxe Changer un type de données

Les deux fonctions principales de changement de type sont **CAST** et **CONVERT**. Ces deux fonctions sont similaires et ne diffèrent que par leur syntaxe :

`CAST(expression AS type)`

`CONVERT(expression, type)`

En cas d'échec, ces fonctions retournent `NULL`.

Les types de données attendus sont les suivants :

- BINARY
- CHAR
- DATE
- DATETIME
- DECIMAL
- DOUBLE
- FLOAT
- INTEGER
- TIME

Ce sont les types également disponibles dans les opérations `CREATE TABLE`.

Exemple

```
1 SELECT CAST('2030' AS INT); -- Retourne l'entier 2030
2 SELECT CONVERT('2030', INT); -- Retourne l'entier 2030
3
4 SELECT CAST(2030 AS VARCHAR(10)); -- Retourne la chaîne de caractères '2030'
5 SELECT CAST(2030 AS VARCHAR(3)); -- Retourne la chaîne de caractères '203', tronquée car la
   taille maximum est 3
6
7 SELECT CAST('2030-05-15' AS DATE); -- Retourne la date 2030-05-15
8 SELECT CAST('15-05-2030' AS DATE); -- Retourne NULL, le SGBD ne sait pas convertir nativement
   cette date
9 SELECT CAST('05-15-2030' AS DATE); -- Retourne NULL, le SGBD ne sait pas convertir nativement
   cette date
10
```

```
11 SELECT CAST('2030-05-15' AS INT); -- Retourne 2030, le SGBD essaye de convertir ce qu'il peut en nombre, puis s'arrête au premier caractère non numérique
```

Remarque

Le SGBD essaye de convertir les données comme il le peut. Parfois, certaines données seront tronquées ou incorrectes.

Dans les exemples ci-dessus, les formats de date 15-05-2030 et 05-15-2030 semblent tout à fait compréhensibles par un humain, mais ils sont ambigus pour le SGBD. Par conséquent, le SGBD renvoie NULL.

Par contre, 2030-05-15 est converti partiellement en INT et donne la valeur 2030.

Fonctions de formatage

MariaDB fournit également des fonctions de formatage. Il s'agit en pratique de fonctions convertissant une donnée source en texte formaté.

- `FORMAT` prend en paramètres un nombre et un nombre de chiffres après la virgule, et retourne une chaîne de caractères contenant le nombre formaté, avec un arrondi si le nombre est tronqué.
- `DATE_FORMAT` prend en paramètres une date et un format de date, et retourne une chaîne de caractères contenant la date formatée.

À noter que ces deux fonctions renvoient du texte. En apparence, la fonction `FORMAT` pourrait être confondue avec la fonction `ROUND`, mais le type renvoyé n'est pas le même.

Exemple

```
1 SELECT FORMAT(123.456, 1); -- Retourne '123.5'
2 SELECT DATE_FORMAT('2020-05-15', '%d/%m/%Y'); -- Retourne '15/05/2020'
```

Remarque

La fonction `STR_TO_DATE` vue précédemment pour les dates est également une fonction de conversion. Elle est similaire à un `CAST (xxx AS DATE)` ou `CONVERT (xxx, DATE)`.

Toutefois, il est préférable d'utiliser `STR_TO_DATE`, car elle permet de spécifier un format de date bien précis.

Syntaxe **À retenir**

- Les fonctions principales de conversion de données sont `CAST` et `CONVERT`. Ces fonctions permettent de passer de tout type à un autre.
- MariaDB fournit également deux fonctions (`FORMAT` et `DATE_FORMAT`) permettant de convertir des données vers du texte en précisant un formatage.

Complément

Documentation CAST¹

Documentation CONVERT²

Documentation FORMAT³

1 <https://mariadb.com/kb/en/cast/>

2 <https://mariadb.com/kb/en/convert/>

3 <https://mariadb.com/kb/en/format/>

Documentation DATE_FORMAT¹

VII. Exercice : Appliquez la notion

Question

[solution n°4 p.21]

Dans l'habillement, il est courant de mettre le numéro de l'année avec le code produit, afin d'indiquer clairement qu'un produit est de la gamme 2020, un autre de la gamme 2019, etc.

Par exemple, si un produit a le code ABC123, il sera étiqueté 2020_ABC123.

Sachant que, pour l'exemple, la requête suivante donne le code produit :

```
1 SELECT 'ABC123';
```

Écrivez une requête renvoyant le code produit final.

Indice :

Pour concaténer deux chaînes de caractères, il faut utiliser CONCAT.

Indice :

La requête suivante donne l'année :

```
1 SELECT YEAR(NOW());
```

Indice :

YEAR(NOW()) renvoie un nombre, il faut utiliser CAST ou CONVERT pour le convertir en VARCHAR(4).

VIII. Fonctions d'agrégation

Objectifs

- Comprendre ce qu'est une agrégation
- Effectuer des opérations d'agrégation

Mise en situation

Les fonctions vues jusqu'à présent retournaient un résultat par valeur traitée. Par exemple, sur une colonne contenant des valeurs décimales, la fonction ROUND(Colonne) parcourt toutes les valeurs contenues dans la colonne et fait un arrondi sur chaque valeur. S'il y a dix valeurs dans la colonne, il y aura donc dix valeurs retournées.

Cependant, on peut parfois avoir envie de faire des calculs portant sur l'ensemble des valeurs de nos lignes pour retourner un seul résultat. Par exemple, on pourrait vouloir calculer la somme des prix de tous les produits d'une commande. Pour cela, il va falloir utiliser des fonctions d'agrégation.

Définition

Une fonction d'agrégation effectue une opération sur un ensemble de données pour ne renvoyer qu'une seule valeur. Elles prennent en paramètre une colonne d'une table et retournent le résultat de l'opération effectuée sur les valeurs de toutes les lignes de cette colonne.

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

Les fonctions d'agrégation usuelles sont, pour les valeurs numériques :

- SUM : somme
- AVG : moyenne
- COUNT : décompte
- MAX : renvoie la valeur maximale trouvée
- MIN : renvoie la valeur minimale trouvée

Il existe également des fonctions d'agrégations applicables sur d'autres types de données :

- GROUP_CONCAT effectue la concaténation de texte, chaque élément étant séparé par des virgules
- MIN et MAX fonctionnent également sur des chaînes de caractères et sont alors basées sur l'ordre alphabétique

Exemple

Soit une table contenant des produits avec leurs prix respectifs :

```
1 CREATE TABLE Product
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     Code varchar(10) NOT NULL,
5     Price decimal(10,2) NOT NULL
6 )
7 ;
8 INSERT INTO Product VALUES (1, 'CHAISE', 10.2);
9 INSERT INTO Product VALUES (2, 'TABLE', 80);
10 INSERT INTO Product VALUES (3, 'BANC', 40);
```

Les fonctions suivantes vont faire des opérations sur l'ensemble des données de la table (donc les trois produits insérés) :

```
1 SELECT COUNT(Id) FROM Product; -- Retourne le nombre de produits, donc 3
2 SELECT SUM(Price) FROM Product; -- Retourne la somme de tous les prix, donc 130.20
3 SELECT AVG(Price) FROM Product; -- Retourne le prix moyen, donc 43.400000
4 SELECT MAX(Price) FROM Product; -- Retourne le prix maximum, donc 80.00
5 SELECT MIN(Price) FROM Product; -- Retourne le prix minimum, donc 10.20
6 SELECT GROUP_CONCAT(Code) FROM Product; -- Retourne la concaténation de tous les codes
   produit, donc 'CHAISE,TABLE,BANC'
```

Remarque

Ne confondez pas GROUP_CONCAT, qui est une agrégation, avec CONCAT, qui est une fonction de texte.

Dans l'exemple précédent, SELECT GROUP_CONCAT(Code) FROM Product; ne retourne qu'une seule valeur, composée de la concaténation de tous les codes, alors que SELECT CONCAT(Code) FROM Product; retourne autant de valeurs que de produits (qui ne contiennent que le code produit, étant donné qu'il n'y a qu'un seul paramètre).

Syntaxe À retenir

- Les fonctions d'agrégation transforment un ensemble de données en une valeur unique.
- Les fonctions les plus courantes sont COUNT (décompte), SUM (somme) et GROUP_CONCAT (concaténation).

Complément

Documentation MariaDB¹

IX. Exercice : Appliquez la notion

Question

[solution n°5 p.21]

Un magasin dispose d'une base de données listant les ventes effectuées. Voici la structure de la table :

```
1 CREATE TABLE Ventes (
2   Id INT PRIMARY KEY AUTO_INCREMENT,
3   NumeroFacture VARCHAR(20) NOT NULL,
4   MontantVente DECIMAL(11, 2) NOT NULL DEFAULT 0,
5   date DATE
6 );
```

Écrivez une requête renvoyant la somme des ventes, le montant moyen et le montant maximum.

Indice :

Les fonctions d'agrégation correspondantes sont SUM, AVG et MAX.

X. Expressions conditionnelles

Objectifs

- Utiliser une structure conditionnelle dans une requête
- Manipuler des expressions booléennes

Mise en situation

Il est souvent nécessaire d'effectuer un traitement conditionnel sur des données, selon une condition pouvant varier à chaque ligne d'une requête. Par exemple, un vendeur offre souvent les frais de port à partir d'un certain montant de ventes. Il serait possible de résoudre ce problème en écrivant deux requêtes : une pour récupérer les commandes ayant atteint le seuil, et une seconde pour les autres. Mais il est possible d'être plus efficace et de tout faire dans la même requête grâce aux expressions conditionnelles.

Définition **Expressions conditionnelles**

Une expression conditionnelle permet de retourner une valeur par cas possible. Elle se rapproche d'un switch d'autres langages de programmation, mais ne fait que retourner des valeurs.

À l'intérieur d'une requête SELECT, il est possible d'utiliser la syntaxe conditionnelle suivante :

```
1 CASE
2   WHEN Condition1 THEN Valeur1
3   WHEN Condition2 THEN Valeur2
4   (...)
5   ELSE ValeurSinon
6 END
```

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

Exemple

Reprenons notre exemple précédent et admettons que nous ayons une table des ventes sous cette forme :

```
1 CREATE TABLE Sale
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     Price decimal(10,2) NOT NULL
5 );
6
7 INSERT INTO Sale VALUES (1, 10);
8 INSERT INTO Sale VALUES (2, 100);
9 INSERT INTO Sale VALUES (3, 40);
```

Dans cet exemple, le tarif de livraison est de 2 euros pour les livraisons de moins de 20 euros, 5 euros pour celles de 20 à 50 euros, et offerte au-delà. Pour calculer les frais de port de chaque commande, il suffit d'utiliser une expression conditionnelle.

```
1 SELECT
2     Id,
3     Price,
4     CASE
5         WHEN Price >= 50 THEN 0 -- Si le prix est supérieur à 50, on retourne 0
6         WHEN Price >= 20 THEN 5 -- Sinon, si le prix est supérieur à 20, on retourne 5
7         ELSE 2 -- Sinon on retourne 2
8     END AS Delivery -- On donne un alias à cette colonne pour pouvoir la manipuler simplement
9 FROM Sale;
```

Le résultat sera le suivant :

Id	Price	Delivery
1	10.00	2
2	100.00	0
3	40.00	5

Remarque

Dans une instruction `CASE WHEN`, chaque condition n'est jouée que si les précédentes sont fausses. C'est pour cette raison que, dans l'exemple précédent, il n'y a pas besoin de préciser `Price >= 20 AND Price < 50` dans la seconde condition : les valeurs supérieures ou égales à 50 sont déjà filtrées par la première condition.

Cela signifie également que l'ordre des conditions est important.

Définition Booléens

Une expression booléenne est une expression qui s'évalue en vrai ou faux : jusqu'à présent, le terme de « condition » a été employé, mais il est plus juste de parler d'expression booléenne. Par extension, une fonction booléenne est une fonction qui retourne les valeurs « vrai » ou « faux ».

Les expressions booléennes et les fonctions booléennes peuvent être utilisées dans la clause `WHEN` d'une expression conditionnelle. Par exemple, la fonction `ISNULL (Colonne)` est une fonction booléenne qui renvoie 1 (vrai) si la valeur d'une ligne de la colonne est nulle, 0 (faux) sinon. Il est possible de l'utiliser dans un `CASE` pour retourner une valeur en cas de `NULL`.

Remarque

ISNULL (Colonne) est l'équivalent en fonction de l'expression booléenne Colonne IS NULL.

Pour donner une valeur par défaut à une colonne nulle, il est également possible d'utiliser la fonction IFNULL (Colonne, Défaut). Cette fonction teste la valeur d'une ligne de la colonne : si elle est nulle, la fonction renvoie la valeur par défaut donnée en second argument, sinon elle renvoie la valeur de la colonne. Cette fonction permet de remplacer une expression conditionnelle. C'est l'équivalent de :

```
1 CASE
2   WHEN Colonne IS NULL THEN Defaut
3   ELSE Colonne
4 END
```

Syntaxe À retenir

- Pour effectuer un test sur une valeur, il faut employer l'instruction CASE WHEN ... THEN ELSE ... END : ce sont des expressions conditionnelles.
- Les fonctions booléennes, comme ISNULL, peuvent être utilisées dans la clause WHEN.
- Il est parfois possible de remplacer une expression conditionnelle par une fonction comme IFNULL.

Complément

Documentation ISNULL¹

Documentation IFNULL²³

XI. Exercice : Appliquez la notion

Question

[solution n°6 p.21]

Un magasin dispose d'une base de données listant les ventes effectuées.

Cette base contient une table listant les vendeurs, leur total de vente et l'objectif qu'ils avaient à réaliser.

```
1 CREATE TABLE Performances (
2   Id INT PRIMARY KEY AUTO_INCREMENT,
3   NomVendeur VARCHAR(20) NOT NULL,
4   TotalVentes DECIMAL(11, 2) NOT NULL DEFAULT 0,
5   ObjectifAnnuel DECIMAL(11, 2) NOT NULL DEFAULT 0
6 );
```

Écrivez une requête renvoyant le nom du vendeur, puis Oui ou Non si l'objectif a été atteint ou non.

Indice :

Il faut employer une condition CASE WHEN en effectuant un test sur TotalVentes et ObjectifAnnuel.

Indice :

N'hésitez pas à espacer la requête sur plusieurs lignes pour la rendre plus lisible.

XII. Essentiel

1 <https://mariadb.com/kb/en/isnull/>

2 <https://mariadb.com/kb/en/ifnull/>

3

XIII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°7 p.21]

Exercice

Pour concaténer du texte dans MariaDB, il faut écrire...

- ☐ SELECT 'A' + 'B'
- ☐ SELECT CONCAT('A', 'B')
- ☐ SELECT 'A' & 'B'

Exercice

Quelle est la différence entre les opérateurs X / Y et $X \text{ DIV } Y$?

- ☐ Elles sont équivalentes : `DIV` est synonyme de `/`
- ☐ `DIV` n'existe pas
- ☐ Elles sont différentes : l'une donne une division entière, l'autre une division à virgule flottante

Exercice

Pour stocker une date dans une base de données, il faut utiliser...

- ☐ Une colonne de type `Date`
- ☐ Une colonne de type `varchar` formatée en année-mois-jour
- ☐ Une colonne de type `varchar` formatée selon le format régional

Exercice

Pour obtenir la date de la veille, il faut écrire...

- ☐ SELECT NOW() - 1
- ☐ SELECT NOW() - 1 DAY
- ☐ SELECT DATE_ADD(NOW(), INTERVAL -1 DAY)

Exercice

Les conversions de type...

- ☐ Se font via les fonctions `CAST` ou `CONVERT`
- ☐ Sont faites automatiquement par la base de données
- ☐ Inutiles

Exercice

Pour convertir un texte en date dans MariaDB, la meilleure façon de procéder est...

- ☐ D'effectuer un `CAST (MaDate AS DATE)`
- ☐ D'extraire du texte les valeurs d'année, de mois et de jour, puis de créer la date via la fonction `MAKEDATE`
- ☐ D'utiliser la fonction `STR_TO_DATE` en précisant le format

Exercice

Une fonction d'agrégation permet...

- ☐ D'effectuer une opération unitaire sur chaque ligne, comme une addition, une multiplication...
- ☐ D'effectuer une opération pour transformer un ensemble en un autre, par exemple inverser chaque valeur
- ☐ D'obtenir une valeur unique calculée depuis un ensemble comme une somme ou un décompte

Exercice

La fonction `GROUP_CONCAT` permet...

- ☐ De concaténer deux chaînes entre elles
- ☐ De concaténer toutes les valeurs d'un ensemble
- ☐ De concaténer deux chaînes entre elles avec un séparateur

Exercice

Il est possible d'effectuer une opération conditionnelle avec...

- ☐ `IF ... THEN ... ELSE ... END IF`
- ☐ `CASE WHEN ... THEN ... ELSE ... END`
- ☐ `SWITCH CASE ... : ... BREAK ;`

Exercice

Dans MariaDB, `ISNULL` permet de...

- ☐ Tester si une valeur est nulle : elle renvoie « vrai » ou « faux »
- ☐ Si une valeur est nulle, la remplacer par une valeur par défaut
- ☐ Placer une valeur à `NULL`

B. Exercice : Défi

Un magasin de vente possède une table d'analyse des performances de ces vendeurs. La table est la suivante :

```
1 CREATE TABLE AnalyseVentes
2 (
3     Id int NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     NomVendeur varchar(50) NOT NULL,
5     PrenomVendeur varchar(50) NOT NULL,
6     Exercice int,
7     TotalVentes decimal(15,2),
8     ObjectifAnnuel decimal(15,2)
9 )
10 ;
```

Cette table peut être alimentée avec les données suivantes :

```
1 INSERT INTO AnalyseVentes VALUES (1, 'Dupont', 'Jean', 2020, 157007.50, 150000);
2 INSERT INTO AnalyseVentes VALUES (2, 'Dupond', 'Jeanne', 2020, 188045.14, 160000);
3 INSERT INTO AnalyseVentes VALUES (3, 'Duperre', 'Jeremy', 2020, 138790.8, 140000);
```

Question

[solution n°8 p.24]

Écrivez une requête permettant d'obtenir :

- Dans une seule colonne, le prénom suivi d'un espace, puis le nom en majuscules. Par exemple « Jean DUPONT »
- La date de début de l'exercice, c'est-à-dire le 1^{er} janvier de l'année donnée dans la colonne Exercice
- La date de fin de l'exercice, c'est-à-dire le 31 décembre de l'année donnée dans la colonne Exercice
- Le total des ventes
- L'écart entre le total des ventes et l'objectif annuel : cet écart doit être positif si le total des ventes a dépassé l'objectif
- Un texte indiquant `Oui` si l'objectif a été atteint, ou `Non` dans le cas contraire

Indice :

Il est plus simple d'écrire la requête petit à petit, en testant après chaque colonne.

Indice :

Pour les dates, la fonction `MAKEDATE` permet d'obtenir très simplement la date de début. Attention par contre à la date de fin, ce n'est pas nécessairement le 365^{ème} jour de l'année !

Une petite astuce dans ce cas est de calculer la date du 1^{er} janvier de l'année suivante, puis de retrancher un jour.

Indice :

Ne vous laissez pas intimider par la formulation pour l'écart des ventes. Il s'agit d'une simple soustraction.

Indice :

Pour le `Oui` / `Non`, il faut employer une expression conditionnelle.

Par contre, il n'est pas possible de réutiliser le résultat du calcul de la colonne précédente, il faut refaire le calcul dans la condition.

Solutions des exercices

p. 6 Solution n°1

Il y a deux possibilités : soit en utilisant `CONCAT`, soit en utilisant `CONCAT_WS`.

```
1 SELECT CONCAT(Prenom, ' ', Nom) FROM Utilisateurs;  
2 SELECT CONCAT_WS(' ', Prenom, Nom) FROM Utilisateurs;
```

L'intérêt de `CONCAT_WS` serait plus apparent avec un plus grand nombre de colonnes, car il n'y a besoin de préciser le séparateur qu'une seule fois.

p. 7 Solution n°2

```
1 SELECT UPPER(CONCAT(LEFT(Prenom, 1), LEFT(Nom, 1), RIGHT(Nom, 1))) FROM Utilisateurs;
```

p. 9 Solution n°3

```
1 SELECT DAY(NOW()), MONTH(NOW()), YEAR(NOW());
```

p. 12 Solution n°4

```
1 SELECT CONCAT(CAST(YEAR(NOW()) AS VARCHAR(4)), '_', 'ABC123')
```

p. 14 Solution n°5

```
1 SELECT SUM(MontantVente), AVG(MontantVente), MAX(MontantVente) FROM Ventes
```

p. 16 Solution n°6

```
1 SELECT  
2     NomVendeur,  
3     CASE  
4         WHEN TotalVentes > ObjectifAnnuel THEN 'Oui'  
5         ELSE 'Non'  
6     END AS ObjectifAtteint  
7 FROM Performances
```

Exercice p. 17 Solution n°7**Exercice**

Pour concaténer du texte dans MariaDB, il faut écrire...

- ☐ `SELECT 'A' + 'B'`
+ est un opérateur numérique.
- ☒ `SELECT CONCAT('A', 'B')`
- ☐ `SELECT 'A' & 'B'`
& est un opérateur logique.

Q La fonction de texte permettant de faire la concaténation est `CONCAT`. Il existe également `CONCAT_WS`, qui permet de rajouter un séparateur entre les différentes valeurs.

Exercice

Quelle est la différence entre les opérateurs `X / Y` et `X DIV Y` ?

- ☐ Elles sont équivalentes : `DIV` est synonyme de `/`
- ☐ `DIV` n'existe pas
- ☒ Elles sont différentes : l'une donne une division entière, l'autre une division à virgule flottante

Q `/` est l'opérateur de division flottante, `DIV` est l'opérateur de division entière (et `MOD` permet de récupérer le reste d'une division entière).

Exercice

Pour stocker une date dans une base de données, il faut utiliser...

- ☒ Une colonne de type `Date`
- ☐ Une colonne de type `varchar` formatée en année-mois-jour
- ☐ Une colonne de type `varchar` formatée selon le format régional

Q Il existe plusieurs formats de date, comme `DATE`, `DATETIME`... selon les besoins. Dans le cas d'une date, le type `DATE` suffit. Dans tous les cas, il convient d'éviter à tout prix les formats textes pour des dates.

Exercice

Pour obtenir la date de la veille, il faut écrire...

- ☐ `SELECT NOW() - 1`
- ☐ `SELECT NOW() - 1 DAY`
- ☒ `SELECT DATE_ADD(NOW(), INTERVAL -1 DAY)`

Q La fonction `DATE_ADD` permet de rajouter ou d'enlever des jours, mois... à une date.

Exercice


Les conversions de type...

- ☒ Se font via les fonctions `CAST` ou `CONVERT`
- ☐ Sont faites automatiquement par la base de données
- ☐ Inutiles

Q Il faut utiliser les fonctions `CAST` ou `CONVERT` en précisant le type souhaité. Le SGBD effectue automatiquement certaines conversions : cependant, il est préférable de les faire explicitement.


Exercice

Pour convertir un texte en date dans MariaDB, la meilleure façon de procéder est...

- ☐ D'effectuer un `CAST (MaDate AS DATE)`
- ☐ D'extraire du texte les valeurs d'année, de mois et de jour, puis de créer la date via la fonction `MAKEDATE`
- ☒ D'utiliser la fonction `STR_TO_DATE` en précisant le format
-  La fonction `STR_TO_DATE` est faite pour ça. Il est toujours préférable d'utiliser une fonction de date dédiée plutôt que de chercher à manipuler des dates au format texte.


Exercice

Une fonction d'agrégation permet...

- ☐ D'effectuer une opération unitaire sur chaque ligne, comme une addition, une multiplication...
- ☐ D'effectuer une opération pour transformer un ensemble en un autre, par exemple inverser chaque valeur
- ☒ D'obtenir une valeur unique calculée depuis un ensemble comme une somme ou un décompte
-  Les fonctions d'agrégation prennent en entrée un ensemble et fournissent une valeur unique.


Exercice

La fonction `GROUP_CONCAT` permet...

- ☐ De concaténer deux chaînes entre elles
- ☒ De concaténer toutes les valeurs d'un ensemble
- ☐ De concaténer deux chaînes entre elles avec un séparateur
-  Contrairement à `CONCAT` et `CONCAT_WS` qui sont des fonctions de manipulation de chaînes, `GROUP_CONCAT` est une fonction d'agrégation.


Exercice

Il est possible d'effectuer une opération conditionnelle avec...

- ☐ `IF ... THEN ... ELSE ... END IF`
- ☒ `CASE WHEN ... THEN ... ELSE ... END`
- ☐ `SWITCH CASE ... : ... BREAK ;`
-  L'instruction conditionnelle en SQL est `CASE WHEN`.

Exercice

Dans MariaDB, `ISNULL` permet de...

- ☒ Tester si une valeur est nulle : elle renvoie « vrai » ou « faux »
- ☐ Si une valeur est nulle, la remplacer par une valeur par défaut
- ☐ Placer une valeur à `NULL`
-  Dans MariaDB, `ISNULL` renvoie « vrai » si le paramètre est à `NULL`, et « faux » sinon.
Dans d'autres SGBD, `ISNULL` peut avoir une fonction différente ! Il faut se référer à la documentation du SGBD le cas échéant.

p. 19 Solution n°8

Pour le libellé de chaque vendeur, on utilise la fonction `CONCAT` pour concaténer le nom et le prénom. La fonction `UPPER` permet de mettre le nom en majuscules.

Pour la date de début de l'exercice, on crée une date en utilisant la fonction `MAKEDATE`. L'année de l'exercice est celle contenue dans la table, et le mois est 1 (janvier).

Pour la date de fin de l'exercice, il n'est pas possible de simplement ajouter 365 jours à la date de début : certaines années sont bissextiles. L'approche est donc plutôt de récupérer le 1^{er} janvier de l'année suivante (donc `MAKEDATE(Exercice + 1, 1)`), puis de retrancher un jour grâce à `DATE_ADD`.

Le total des ventes est simplement une des colonnes de la table, et l'écart est sa soustraction avec la colonne de l'objectif annuel.

Enfin, pour afficher si l'objectif a été atteint ou non, on utilise une structure conditionnelle en comparant le total des ventes à l'objectif.

La requête finale est donc :

```

1 SELECT
2   CONCAT(PrenomVendeur, ' ', UPPER(NomVendeur)) AS Vendeur,
3   MAKEDATE(Exercice, 1) AS DateDebutExercice,
4   DATE_ADD(MAKEDATE(Exercice + 1, 1), INTERVAL -1 DAY) AS DateFinExercice,
5   TotalVentes,
6   ObjectifAnnuel,
7   Calcul de l'écart avec ABS pour garantir qu'il est toujours positif
8   ABS(TotalVentes - ObjectifAnnuel) AS EcartPositif,
9   Indicateur d'objectif atteint
10  CASE
11  WHEN TotalVentes >= ObjectifAnnuel THEN 'Oui'
12  ELSE 'Non'
13  END AS ObjectifAtteint
14  FROM
15  AnalyseVentes;
```