Bases de donnés et SQL

2021

TECH & CODE FACTORY

Détails du cours

But du cours: Le cours a pour but de donner aux étudiants une compréhension globale de l'utilisation de bases de données dans notre monde aujourd'hui ainsi que les compétences pour savoir manipuler des bases de données relationnelles, notamment grâce au langage SQL.

Heures de cours: 55h - 11 sessions de 5h

c.duquesne@prof-webschoolfactory.fr

Modalités d’évaluation

Cours

Projet

Référez vous au syllabus

Planning théorique du cours

Cours

1. Introduction aux bases de donnés, installation de MySQL, Introduction à SQL
2. Introduction à SQL
3. Opération CRUD en SQL, contraintes en SQL, Diagrammes d'entité
4. Joindre des tables en SQL, théorème ACID
5. Subqueries, transactions, optimisation de queries, views, procedures...
6. Utilisation et écriture de fonctions en SQL
7. Design de bases de données relationnelles
8. Initiation à MongoDB
9. Projet
10. Projet
11. Evaluation

Cours 1

15 - 9 - 2021

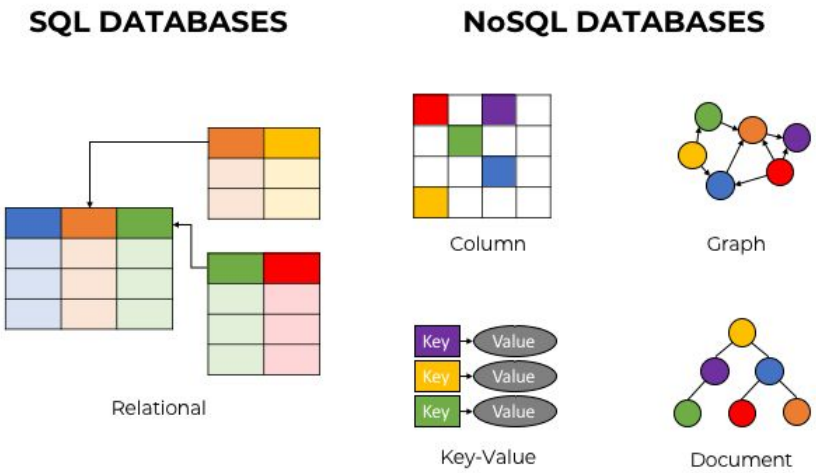
Introduction

Qu’est ce qu’une base de donnés ?

Une base de données est un ensemble d'informations qui est **organisé** de manière à être facilement accessible, géré et mis à jour. Elle est utilisée par les organisations comme méthode de **stockage**, de **gestion** et de **récupération** de l’informations. Les premières bases de donnés informatiques commencent à apparaître environ aux alentours des années 1960-1970.

Quels sont les différents types de bases de donnés ?

Cours



La façon dont les données sont organisées détermine souvent le **type** de base de données utilisé. Les bases de données sont largement divisées en deux grands types ou catégories **les bases de données relationnelles SQL** et les **bases de donnés noSQL**.

Qu’est ce que le SQL ?

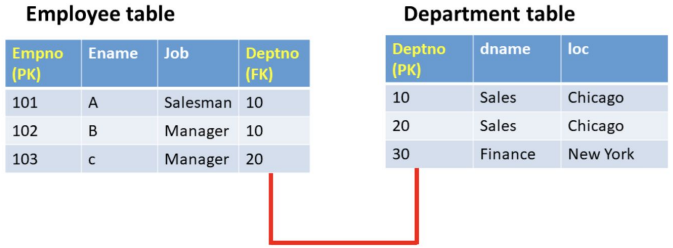
SQL est l'abréviation de **Structured Query Language.** Le **langage** SQL est utilisé pour **communiquer avec une base de données**. Selon l'ANSI (American National Standards Institute), il s'agit du langage standard pour les systèmes de gestion de bases de données **relationnelles**. Les instructions SQL sont utilisées pour effectuer des tâches telles que la mise à jour de données dans une base de données ou la récupération de données dans une base de données.

SQL a été créé au début des années 1970 chez IBM comme méthode d'accès à leur système de base de données. Ce fut ensuite Orcale qui a mis sur le marché la première base de données relationnelle commerciale en 1979 (et qui fut suivie par de nombreuses autres entreprises ensuite). C’est donc dans les années 1980 et 1990, que les bases de données relationnelles sont devenues de plus en plus dominantes sur le marché, car elles offraient un système riche pour rendre toute requête efficace.

Qu’est ce que sont les bases de données relationnelles ?

Cours

Une base de données relationnelle est un type de base de données où les données sont **liées** à d'autres informations au sein des bases de données. Les bases de données relationnelles sont composées d’un ensemble de **tables** qui peuvent être accessibles et reconstruites de différentes manières, sans qu'il soit nécessaire de réarranger ces tables de quelque façon que ce soit. Le langage de requête structuré (SQL) est l’interface standard pour une base de données relationnelle.



Employee table

Empno (PK)	Ename	Job	Deptno (FK)
101	A	Salesman	10
102	B	Manager	10
103	c	Manager	20

Department table

Deptno (PK)	dname	loc
10	Sales	Chicago
20	Sales	Chicago
30	Finance	New York

Qui utilise des bases de données relationnelles ?

Microsoft accenture airbnb Uber twitch

TECH & CODE FACTORY

Avantages et inconvénients des bases de données relationnelles

Avantages

- Elles suivent un schéma strict, ce qui signifie que chaque nouvelle entrée doit avoir différents composants qui lui permettent de s'intégrer dans ce modèle préformé. Cela permet aux données d'être prévisibles et faciles à évaluer.
- Elles se conforment aux règles ACID (cf cours futurs) qui assurent l'intégrité des donnés.
- Elles sont bien structurés et ne laissent que peu de place à tout type d'erreur.

Inconvénients

- Il est impossible de mettre à l'échelle horizontalement à cause des schémas stricts. Bien que la mise à l'échelle verticale semble être la réponse évidente, ce n'est pas le cas. La mise à l'échelle verticale a une limite et, à cette époque et à cet âge, les données collectées quotidiennement via Internet sont tout simplement trop volumineuses pour imaginer que la mise à l'échelle verticale fonctionnerait longtemps.

Qu’est ce que sont les bases de données NoSQL ?

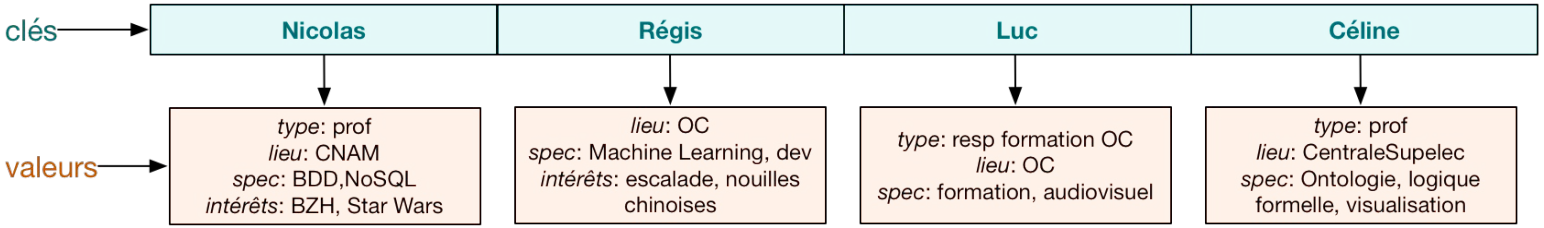
Depuis les années 1970, la base de données relationnelle était l'incontournable référence pour gérer les données d'un système d'information. Toutefois, face au volume, la vitesse et la variété des donnés collectés aujourd’hui, le relationnel peut difficilement lutter et s’adapter. Le NoSQL s'est naturellement imposé dans ce contexte en proposant une nouvelle façon de gérer les données, sans reposer sur le paradigme relationnel, d'où le **"Not Only SQL"**. Cette approche propose de relâcher certaines contraintes lourdes du relationnel pour favoriser la distribution (comme la structure des données, le langage d'interrogation ou la cohérence).

Les bases de donnés clés - valeurs

Cours

Le but de la famille clé-valeur est l'efficacité et la simplicité. Un système clé-valeur agit comme une énorme **table de hachage** distribuée sur le réseau. Tout repose sur le couple **Clé/Valeur**. La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données.

Le fait d'avoir n'importe quoi implique qu'il n'y ait ni schéma, ni structure pour le stockage. D'un point de vue de bases de données, il n'y a pas la possibilité d'exploiter ni de contrôler la structure des données et de fait, pas de langage. En soit ce n'est pas un problème si vous savez ce que vous cherchez (la clé) et que vous manipulez directement la valeur.



Les bases de donnés orientés colonnes

Cours

Traditionnellement, les données sont représentées en ligne, représentant l'ensemble des attributs. Le **stockage orienté colonne** change ce paradigme en se focalisant sur chaque **attribut** et en les distribuant. Il est alors possible de focaliser les requêtes sur une ou plusieurs colonnes, sans avoir à traiter les informations inutiles (les autres colonnes).

Cette solution est très adaptée pour effectuer des traitements sur des colonnes comme les **agrégats** (comptage, moyennes, co-occurences...). D'une manière plus concrète, elle est adaptée à de gros calculs analytiques. Toutefois, cette solution est beaucoup moins appropriée pour la lecture de données spécifiques comme pour les clés/valeurs.

Stockage orienté lignes

id	type	lieu	spec	intérêts
Nicolas	prof	CNAM	BDD, NoSQL	BZH, Star Wars
Régis		OC	Machine Learning, Dev	escalade, nouilles chinoises
Luc	resp formation OC	OC	formation, audiovisuel	
Céline	prof	CentraleSupelec	Ontologie, logique formelle, visualisation	

Stockage orienté colonnes

id	type
Nicolas	prof
Céline	prof
Luc	resp formation OC

id	lieu
Céline	Centrale Supelec
Nicolas	CNAM
Régis	OC
Luc	OC

id	spec
Nicolas	BDD
Nicolas	NoSQL
Régis	Machine Learning
Régis	Dev
Luc	formation
Luc	audiovisuel
Céline	Ontologie
Céline	logique formelle
Céline	visualisation

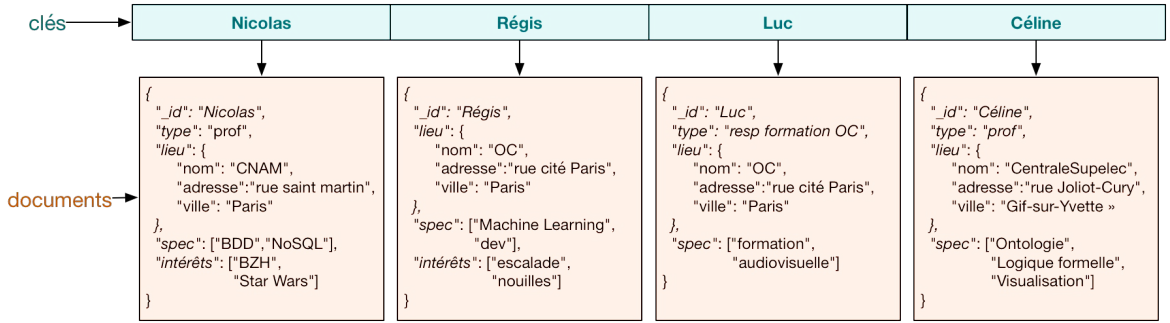
id	intérêts
Nicolas	BZH
Nicolas	Star Wars
Régis	escalade
Régis	nouilles chinoises

Les bases orientées documents

Cours

Les **bases orientées documents** ressemblent sans doute le plus à ce que l'on peut faire dans une base de données classique pour des requêtes complexes. Le but de ce stockage est de manipuler des documents contenant des informations avec une **structure complexe** (types, listes, imbrications). Il repose sur le principe du **clé/valeur**, mais avec une extension sur les champs qui composent ce document.

L'avantage de cette solution est d'avoir une approche structurée de chaque valeur, formant ainsi un document. De fait, ces solutions proposent des langages d'interrogation riches permettant de faire des manipulations complexes sur chaque attribut du document (et sous-documents) comme dans une base de données traditionnelles, tout en passant à l'échelle dans un contexte distribué.



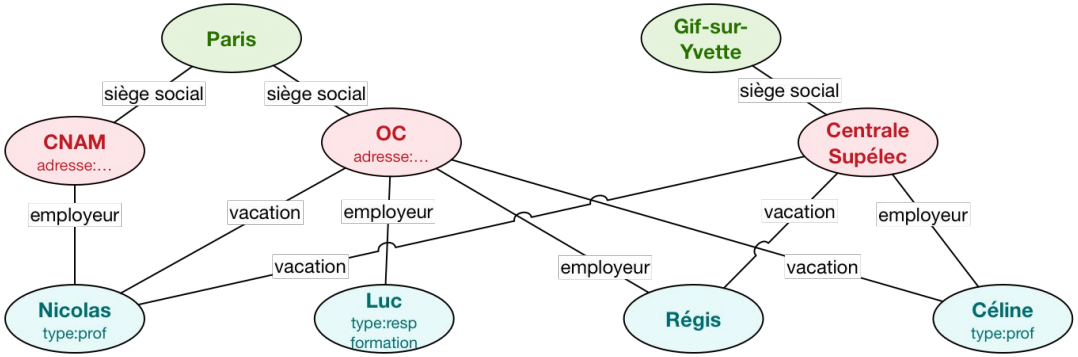
TECH & CODE FACTORY

Les bases de donnés orientés graphes

Cours

Les trois premières familles NoSQL n'adressent pas le problème de corrélations entre les éléments. Prenons l'exemple d'un réseau social : dans certains cas, il devient très complexe de calculer la distance entre deux personnes non directement connectées. Et c'est ce type d'approche que résolvent les bases orientées Graphe.

Dans la **base orientée graphe**, les données stockées sont : **les nœuds, les liens et des propriétés sur ces nœuds et ces liens**. Les requêtes que l'on peut exprimer sont basées sur la gestion de chemins, de propagations, d'agrégations, voire de recommandations. Toutefois, contrairement aux solutions précédentes la distribution des nœuds sur le réseau n'est pas triviale.



TECH & CODE FACTORY

Avantages et inconvénients des bases de données NoSQL

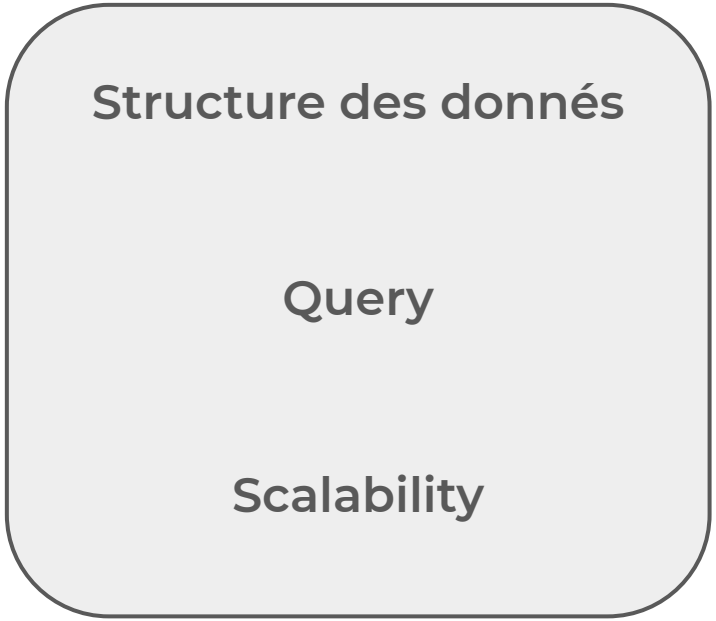
Avantages

- La possibilité d'utiliser autre chose qu'un schéma fixe sous forme de tableaux dont toutes les propriétés sont fixées à l'avance ;
- La possibilité d'avoir un système facilement distribué sur plusieurs serveurs et avec lequel un besoin supplémentaire en stockage ou en montée en charge se traduit simplement par l'ajout de nouveaux serveurs.

Inconvénients

- Le schéma flexible apporte une plus grande liberté au développeur et lui permet de stocker de façon optimale des ensembles de données dont les entrées peuvent être très disparates. Mais en contrepartie, le langage permettant d'effectuer des requêtes vers le système NoSQL est beaucoup moins riche et la complexité intrinsèque de la requête est déplacée du SQL vers la logique de l'application elle-même ;
- En l'absence de relations, il est très difficile de mettre à jour les données car vous devrez mettre à jour chaque détail séparément.

Quand utiliser une BDD SQL ou une BDD NoSQL ?



Pourquoi utiliser des bases de donnés ?

Gérer de grandes quantités de données

Pécision

Facilité de mises à jour des données

Sécurité

Scalability

Intégrité des données

Facilité de recherche des donnés

Qu’est ce qu’un système de gestion de base de données (SGBD) ?

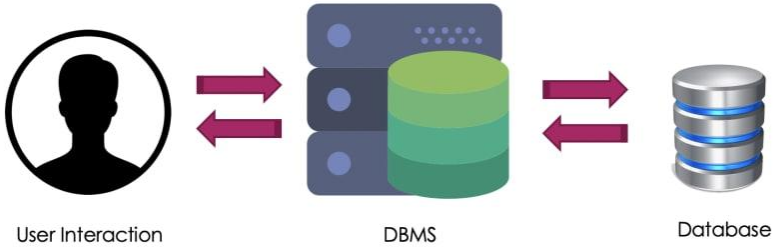
Cours

Le **logiciel** utilisé pour stocker, gérer, interroger et récupérer les données stockées dans une base de données est appelé système de gestion de base de données (SGBD). Le SGBD fournit une interface entre les utilisateurs et les applications et la base de données, ainsi que des fonctions administratives pour gérer le stockage, l'accès et les performances des données.

Il existe des SGBD spécifiques aux bases de données relationnelles appelés **SGBDR.** En voici quelques exemples : *MySQL, Microsoft SQL Server, Oracle, PostgreSQL, MariaDB, etc.*

Il existe également différents SGBD pour les bases de données NoSQL :

- Pour les BDD clé valeurs: *Redis, Memcached, Azure Cosmos DB, SimpleDB*
- Pour les BDD orienté colonne: *BigTable, HBase , Spark SQL , Elasticsearch*
- Pour les BDD orienté document: *MongoDB, CouchBase , DynamoDB , Cassandra*
- Pour les BDD orienté graphs: *Neo4j, OrientDB, FlockDB*



Comment choisir entre les différents SGBD ?

- Licence
- Prix
- OS Supportés
- Features spéciaux
- Ressources
- Scalability
- Documentation
- Facilité d'apprentissage

Les bases de données

Exercices

Remplissez le tableau suivant grâce à ce que l'on vient de voir en cours et grâce à vos propres recherches.

	MySQL	Oracle	MongoDB	Cassandra	MongoDB
Licence					
Prix					
Scalability (Horizontal/Vertical)					
Type de BDD					
Capacité maximum de stockage					

Toutes ces questions peuvent vous être posés en entretien d'embauche !!!

Je vous invite vivement à approfondire les définitions et les questions pour que vous soyez capable d'y répondre avec vos propres mots.

Installation de MySQL

Téléchargement

https://www.mysql.com/ -> downloads -> Mysql community downloads (https://dev.mysql.com/downloads/)

On va télécharger **MySQL community Server** (le SGBD) ET **MySQL Workbench** (l'application visuelle du SGBD)

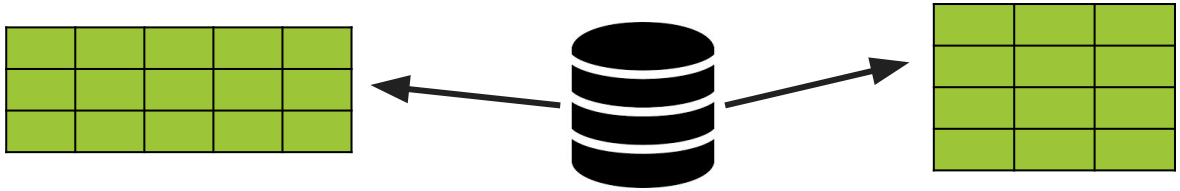
Faire les étapes de l'installation avec les élèves ensemble

Demo: https://www.youtube.com/watch?v=7S_tz1z_5bA

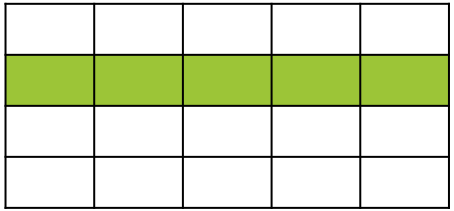
Les bases de SQL

La structure et terminologie de base

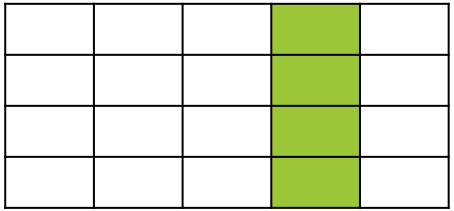
Une **base de données** est une collection de **tables**, également appelées **entités**. *Cours*



Chaque **table** est composée **d'enregistrements**. Chaque enregistrement doit être **unique** et peut être stocké dans n'importe quel ordre dans la table.



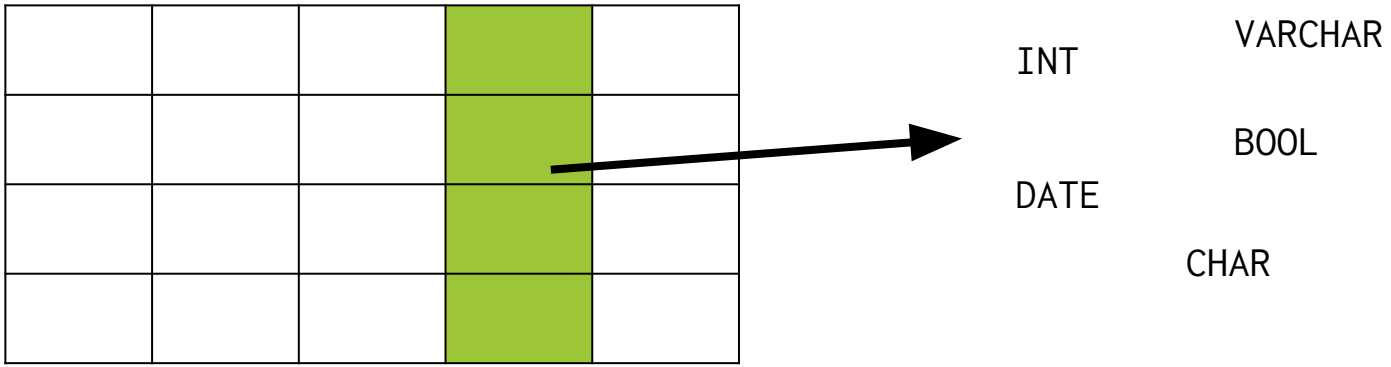
Chaque **enregistrement** est composé de **champs** (qui sont les **colonnes** verticales de la table, également appelées **attributs**).



La structure et terminologie de base

Les **champs** peuvent être de différents **types.**

L'éventail des valeurs autorisées pour un champ est appelé le **domaine** (également appelé **spécification** du champ). Par exemple, un champ relatif aux cartes de crédit peut être limité aux seules valeurs Mastercard, Visa et Amex.



Les types de base en SQL

Types	Description
`VARCHAR(size)`	Une chaîne de longueur VARIABLE (peut contenir des lettres, des chiffres et des caractères spéciaux). Le paramètre size spécifie la longueur maximale de la colonne en caractères - peut être compris entre 0 et 65535.
`DATE`	Une date. Format : AAAA-MM-JJ. La plage prise en charge va de '1000-01-01' à '9999-12-31'.
`BOOL`	Les valeurs nulles sont considérées comme fausses, les valeurs non nulles sont considérées comme vraies.
`FLOAT(p)`	Un nombre à virgule flottante. MySQL utilise la valeur p pour déterminer s'il faut utiliser FLOAT ou DOUBLE pour le type de données résultant. Si p est compris entre 0 et 24, le type de données devient FLOAT(). Si p est compris entre 25 et 53, le type de données devient DOUBLE().
`INT`	Un nombre entier moyen. La plage signée va de -2147483648 à 2147483647. La plage non signée est comprise entre 0 et 4294967295. Le paramètre size spécifie la largeur maximale de l'affichage (qui est de 255).

Les types de base en SQL

Exercices

Complétez le tableau suivant grâce à vos recherches !

Type	Description
`CHAR(size)`	
`DATETIME(fsp)`	
`TINYINT(size)`	
`SMALLINT(size)`	
`TEXT(size)`	
`DECIMAL(size, d)`	
`BINARY(size)`	

Exercice

Exercices

Complétez le tableau suivant grâce à vos recherches

Type	Description
`CHAR(size)`	Une chaîne de longueur FIXE (peut contenir des lettres, des chiffres et des caractères spéciaux). Le paramètre size spécifie la longueur de la colonne en caractères - peut être compris entre 0 et 255. La valeur par défaut est 1
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
`DECIMAL(size, d)`	Un nombre exact à virgule fixe. Le nombre total de chiffres est spécifié dans la taille. Le nombre de chiffres après le point décimal est spécifié dans le paramètre d. Le nombre maximum pour size est 65. Le nombre maximum pour d est 30. La valeur par défaut du paramètre size est 10. La valeur par défaut du paramètre d est 0.
`BINARY(size)`	Equivalent à CHAR(), mais stocke des chaînes d'octets binaires. Le paramètre size spécifie la longueur de la colonne en octets. La valeur par défaut est 1 Les valeurs nulles sont considérées comme fausses, les valeurs non nulles sont considérées comme vraies.

Regardons tout cela sur MySQL Workbench !

- Comment charger un script SQL ?
- Comment run un script SQL ?
- Comment visualiser une table ?
- Comment comment connaître les types de champs ?

MySQL Workbench

Exercices

1. Chargez le fichier *create-database-tech&code.sql*
2. Quel est le nom de la base de donnés ?
3. Combien y a t'il de tables dans cette base de donnés?
4. Combien d'enregistrements y a t'il dans la table *customers*?
5. Combien y a t'il de champs dans la table *customers* ? Quels sont leur noms ? Que représentent ils ? Quel est le type de chaque champ ?
6. Quelle est la valeur minimale et maximale du champ *points* ?
7. Quelles informations nous apprend l'enregistrement numéro 3 ?
8. Répétez les questions 4 et 5 avec les autres tables de la base de donnés

Comment lire une table en SQL ? (SELECT)

La commande SELECT, retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

Syntaxe

```
SELECT nom_du_champ FROM nom_du_tableau;
```

Cette requête SQL va sélectionner (**SELECT**) le **champ** *"nom_du_champ"* provenant (**FROM**) du **tableau** appelé *"nom_du_tableau"*.

Il est également possible de sélectionner plusieurs champs d'une même table ainsi que de sélectionner tous les champs d'une table comme nous allons voir dans les exemples suivants.

SELECT

Exemples

```
SELECT * FROM products;
```

```
SELECT name FROM products;
```

```
SELECT name, unit_price FROM products;
```

SELECT

Exercices

Afficher toute la table contenant les informations client

Afficher la 4e colonne de la table contenant les informations client

Afficher les 3 premières colonnes de la table contenant les informations client

Afficher la première et la dernière colonne de la table contenant les informations client

WHERE

La commande WHERE permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

Syntaxe

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

Opérateurs de comparaison

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

WHERE

Exemples

```
SELECT * FROM customer WHERE client_id = 4;
```

WHERE & Opérateurs de comparaison

Exercices

Affichez toutes les informations des 10 premiers clients

Affichez toutes les informations des clients qui n'habitent pas à Paris

Affichez toutes les informations des clients ayant plus de 1500 points

Affichez toutes les informations des clients de plus de 35 ans

Affichez les numéros de téléphone des clients n'habitant pas à paris

Affichez les informations postales des clients ayant plus de 2000 points

AND - OR - NOT

Les opérateurs logiques AND et OR sont utilisées au sein de la commande WHERE pour combiner des conditions.

Syntaxe

```
SELECT nom_colonnes FROM nom_table
    WHERE condition1 AND condition2
```

```
SELECT nom_colonnes FROM nom_table
    WHERE condition1 OR condition2
```

```
SELECT nom_colonnes FROM nom_table
    WHERE NOT condition1
```

AND - OR - NOT

Ces opérateurs peuvent être combinés ensemble.

Syntaxe

```
SELECT nom_colonnes FROM nom_table

 WHERE condition1 AND (condition2 OR condition3)
```

AND - OR - NOT

Exemples

```
SELECT * FROM products

    WHERE quantity_in_stock > 20 AND unit_price >= 4.50;
```

```
SELECT * FROM products

    WHERE quantity_in_stock <= 10 OR quantity_in_stock >= 1000;
```

```
SELECT * FROM products

    WHERE NOT (quantity_in_stock <= 10 OR quantity_in_stock >= 1000);
```

AND - OR - NOT

Exercices

Affichez toutes les informations des produits ayant un prix entre 10 et 30 euros

Affichez tous les informations clients des clients habitant sur paris et ayant au moins 1000 points

Affichez les informations d'adresse des clients habitant soit dans le 13e arrondissement de paris soit dans le 14e arrondissement de paris

Affichez les téléphones des clients ayant entre 35 ans et 55 ans inclus.

Affichez tous les id des produits ayant un stock supérieur à 20 et inférieur à 100 items.

Affichez toutes les information clients des clients ayant entre 35 ans et 55 ans inclus, qui ont au moins 1000 point

Affichez toutes les information clients des clients ayant entre 35 ans et 55 ans inclus qui n'habitent pas à Paris.

BETWEEN

BETWEEN est utilisé pour sélectionner un intervalle de données dans une requête utilisant WHERE. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. L'exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

Syntaxe

```
SELECT * FROM table

    WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2';
```

BETWEEN

Exemples

```
SELECT * FROM products

    WHERE unit_price BETWEEN 10 AND 30;
```

BETWEEN

Exercices

Affichez les téléphones des clients ayant entre 35 ans et 55 ans

Affichez tous les id des produits ayant un stock entre 20 et 100 items

Affichez toutes les informations des produits ayant un prix entre 20 et 50 euros

Affichez tous les prénoms des clients dont la 1ere lettre de leur prénom est entre a et m.

Est ce que l'opérateur BETWEEN inclus ou exclut les valeurs qu'on lui passe ?

IN

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

Syntaxe

```
SELECT nom_colonne FROM table

WHERE nom_colonne IN ( valeur1, valeur2, valeur3, ... );
```

IN

Exemples

```
SELECT first_name FROM customers

    WHERE first_name = 'Maurice' OR first_name = 'Marie' OR first_name =
'Thimoté';
```

```
SELECT first_name FROM customers

    WHERE first_name IN ( 'Maurice', 'Marie', 'Thimoté' );
```

IN

Exercices

Affichez les informations des clients à paris lyon et grenoble

Affichez les noms des produits ayant un prix de 15.99€, 19.99€ et 29.99€

Affichez les noms et identifiants des produits dont le stock est égal à 0, 1 , 2 ou 3

Affichez les informations des clients habitant à paris rive gauche

IS NULL / IS NOT NULL

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

Syntaxe

```
SELECT * FROM `table`

    WHERE nom_colonne IS NULL;
```

```
SELECT * FROM `table`

    WHERE nom_colonne IS NOT NULL;
```

IS NULL/ IS NOT NULL

Exemples

```
SELECT * FROM products

    WHERE unit_price IS NOT NULL;
```

IS NULL / IS NOT NULL

Exercices

Affichez toutes les informations clients des clients n'ayant pas renseigné un numéro de téléphone

Affichez toutes les informations clients des clients ayant renseigné un numéro de téléphone

Affichez toutes les informations des commandes ayant une date d'envoi

À vous de jouer !

Exercices globaux

Affichez toutes les informations clients des clients entre 35 et 55 ans et qui habitent à paris, lyon ou grenoble

Affichez toutes les informations produits des produits de plus de 25€ ayant un stock positif

Affichez le nom et les numéros de téléphones des clients n'ayant pas renseignés de date de naissance

Affichez le nom et l'adresse des personnes ayant plus de 1500 points et qui ne résident pas à paris.

AS

La commande AS est utilisée pour renommer une colonne ou une table avec un alias.

Un alias n'existe que pour la durée de la requête.

Syntaxe

```
SELECT nom_colonne AS alias_nom_colonne FROM table;
```

AS

Exemples

```
SELECT first_name AS prénom FROM customers;
```

```
SELECT unit_price, quantity_in_stock,  unit_price * quantity_in_stock
AS max_profit FROM products;
```

AS

Exercices

Affichez les informations clients en renommant les noms de colonnes en français

Regroupez les informations d'adresse en une seule colonne se nommant 'adresse' et affichez toutes les adresses des clients

Regroupez le prénom et le nom en une seule colonne se nommant 'nom' et affichez tous les noms des clients

Affichez les identifiants et tous les 'nom' et 'adresse' des clients

Nous payons 5 centimes pour stocker chaque item pour chacun de nos produits, afficher les informations produits ainsi qu'une colonne 'prix_stockage' représentant le prix que nous payons pour stocker tous les items pour chaque produit

LIKE

L'opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Les modèles de recherches sont multiple.

Syntaxe

```
SELECT * FROM table

    WHERE colonne LIKE modele;
```

Wildcards

Il existe 2 wildcards en SQL le caractère “%” et le caractère “_”:

- **% : le symbole pourcentage représente zéro, un ou plusieurs caractères joker.**
- **_ : le symbole underscore représente un seul caractère joker.**

Ils peuvent être utiliser de la manière suivante:

- LIKE ‘%a’ : le caractère “%” est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui se termine par un “a”.
- LIKE ‘a%’ : ce modèle permet de rechercher toutes les lignes de “colonne” qui commence par un “a”.
- LIKE ‘%a%’ : ce modèle est utilisé pour rechercher tous les enregistrement qui utilisent le caractère “a”.
- LIKE ‘pa%on’ : ce modèle permet de rechercher les chaînes qui commence par “pa” et qui se terminent par “on”, comme “pantalon” ou “pardon”.
- LIKE ‘a_c’ : peu utilisé, le caractère “_” (underscore) peut être remplacé par n’importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage “%” peut être remplacé par un nombre incalculable de caractères . Ainsi, ce modèle permet de retourner les lignes “aac”, “abc” ou même “azc”.

Wildcards

Exemples

```
SELECT * FROM customers

    WHERE last_name LIKE “%y%”;
```

Wildcards

Exercices

Trouvez tous les clients possédant un x un y ou z dans leur nom de famille

Affichez tous les produits ne possédant pas le mot “coussin” dans leur nom

Affichez tous les produits ayant le mot “housse” dans leur nom

Trouvez tous les clients possédant deux a dans leur nom prénom.

Trouvez tous les clients ayant un prénom de 6 lettres et qui se finit avec une voyelle.

Trouvez tous les clients ayant donné un numéro de téléphone fixe

Trouvez tous les clients ayant donné un numéro de téléphone portable

REGEXP

Cours

L'opérateur REGEXP est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Les Regular Expression (REGEXP) permettent d'avoir des modèles plus précis qu'avec l'opérateur LIKE. Les Regular Expression ne sont pas qu'utilisés en SQL mais peuvent aussi être utilisés en Javascript, Python, ...

Voici les principales règles des REGEXP:

- '**Azertyuiop**' signifie que Azertyuiop sera cherché dans votre string
- '**^**' signifie en début de string
- '**$**' signifie en fin de string
- '**|**' signifie un or logique et permet de préciser plusieurs pattern de string à chercher
- '**[abcd]**' signfie que ces lettres peuvent être présentes avant/après un pattern de string à chercher
- '**[a-z]**' signifie que toutes cette range de lettres peuvent être présentes avant/après un pattern de string à chercher

Syntaxe

```
SELECT *FROM table

    WHERE colonne REGEXP modele;
```

REGEXP

Exemples

```
SELECT * FROM customers
    WHERE last_name REGEXP “de”;
```

```
SELECT * FROM customers
    WHERE last_name REGEXP “de|la”;
```

```
SELECT * FROM customers
    WHERE last_name REGEXP “^de|la$”;
```

```
SELECT * FROM customers
    WHERE last_name REGEXP “^de[abcd]”;
```

```
SELECT * FROM customers
    WHERE last_name REGEXP “^de[a-g]”;
```

REGEXP

Exercices

Trouvez tous les clients possédant un x un y ou z dans leur nom de famille

Affichez tous les produits ayant le mot “housse” ou “coton” dans leur nom

Affichez tous les clients dont le nom de famille finit par une voyelle

Trouvez tous les clients ayant donné un numéro de téléphone fixe

Trouvez tous les clients ayant donné un numéro de téléphone portable

À vous de jouer !

Exercices globaux

Téléchargez le fichier pays.sql puis importez le dans votre MySQL workbench.
Exécutez le fichier pour créer la base de donnés puis passez aux exercices suivants:

1. Trouvez les pays qui commencent par Y
2. Trouvez les pays qui se terminent par y
3. Trouvez les pays qui contiennent la lettre x
4. Trouvez les pays qui se terminent par land
5. Trouvez les pays dont le nom commence par C et se termine par ia
6. Trouvez les pays dont le nom comporte la lettre oo
7. Trouvez les pays dont le nom contient trois a ou plus
8. Trouvez les pays dont le deuxième caractère est "t".
9. Trouvez les pays qui ont exactement quatre caractères.
10. Trouvez les pays ayant le même nom que leur capitale
11. Trouvez les pays qui ont plus de 10 millions d'habitants.
12. Trouvez les pays qui se terminent par "land" et qui ont plus de 5 millions d'habitants.
13. Trouvez les pays qui ont moins d'un million d'hectares de terres.
14. Calculez une nouvelle colonne qui est la densité d'habitants : nombre d'habitants par hectare de terre.
15. Trouvez les pays qui ont une densité d'habitants comprise entre 500 et 5000 et qui contiennent soit les lettres 'ai', 'en', 'am', 'al' ou 'ag'.

ORDER BY

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

Syntaxe

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

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

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

ORDER BY

Exemples

```
SELECT first_name, last_name

FROM customers

ORDER BY last_name;
```

```
SELECT first_name, last_name

FROM customers

ORDER BY last_name DESC;
```

ORDER BY

Exercices

1. Affichez tous les produits dans leur ordre alphabétique
2. Affichez tous les informations produits par prix croissant
3. Affichez tous les informations produits par prix décroissant
4. Affichez toutes les informations clients par nombre de points décroissants
5. Affichez toutes les informations clients par ville en ordre alphabétique puis par nom en ordre alphabétique

SUM()

Dans le langage SQL, la fonction d'agrégation SUM() permet de calculer la somme totale d'une colonne contenant des valeurs numériques. Cette fonction ne fonction que sur des colonnes de types numériques (INT, FLOAT ...) et n'additionne pas les valeurs NULL

Syntaxe

```
SELECT SUM(nom_colonne)

FROM table
```

Il est possible de filtrer les enregistrements avec la commande WHERE pour ne calculer la somme que des éléments souhaités.

SUM()

Exemples

```
SELECT SUM(points)

FROM customers
```

```
SELECT SUM(points)

FROM customers

WHERE city = ‘paris’
```

COUNT()

En SQL, la fonction d'agrégation COUNT() permet de compter le nombre d'enregistrement dans une table. Il est aussi possible de connaître le nombre d'enregistrement sur une colonne en particulier. Les enregistrements qui possèdent la valeur nul ne seront pas comptabilisé. Enfin, il est également possible de compter le nombre d'enregistrement distinct pour une colonne. La fonction ne comptabilise pas les doublons pour une colonne choisie.

Syntaxe

```
SELECT COUNT(*) FROM table;

SELECT COUNT(nom_colonne) FROM table;

SELECT COUNT(DISTINCT nom_colonne) FROM table;
```

COUNT()

Exemples

```
SELECT COUNT(*) FROM customers
```

```
SELECT COUNT(birth_date) FROM customers
```

```
SELECT COUNT(DISTINCT city) FROM customers
```

MAX()

La fonction d'agrégation MAX() permet de retourner la valeur maximale d'une colonne dans un set d'enregistrement. La fonction peut s'appliquer à des données numériques ou alphanumériques.

Syntaxe

```
SELECT MAX(nom_colonne) FROM table
```

MAX()

Exemples

```
SELECT MAX(points) FROM customers
```

```
SELECT first_name, last_name, MAX(points) FROM customers
```

MIN()

La fonction d'agrégation MIN() de SQL permet de retourner la plus petite valeur d'une colonne sélectionnée. Cette fonction s'applique aussi bien à des données numériques qu'à des données alphanumériques.

Syntaxe

```
SELECT MIN(nom_colonne) FROM table
```

MIN()

Exemples

```
SELECT MIN(points) FROM customers
```

```
SELECT first_name, last_name, MIN(points) FROM customers
```

AVG()

La fonction d'agrégation AVG() dans le langage SQL permet de calculer une valeur moyenne sur un ensemble d'enregistrement de type numérique et non nul.

Syntaxe

```
SELECT AVG(nom_colonne) FROM nom_table
```

AVG()

Exemples

```
SELECT AVG(points) FROM customers;
```

SUM(), COUNT(), MAX(), MIN(), AVG()

Exercices

1. Combien qu'a t'il d'articles en stock ?
2. Combien d'articles en lin y a t'il en stock ?
3. Combien d'articles en coton y a t'il en stock ?
4. Combien de personnes ont renseigné un numéro de téléphone ?
5. Dans combien de départements avons nous des clients ?
6. Combien de prix différents proposons nous pour nos articles ?
7. Quel est l'article le plus cher que nous vendons ?
8. Quel est l'article le moins cher que nous vendons ?
9. Quel est l'article dont nous avons le moins en stock ?
10. Quel est l'article dont nous avons le plus en stock ?
11. Quel est le prix moyen de nos articles ?
12. Quel est le stock moyen de nos articles ?

CASE

Cours

La commande “CASE … WHEN …” permet d’utiliser des conditions de type “si / sinon” (cf. if / else) similaire à un langage de programmation pour retourner un résultat disponible entre plusieurs possibilités.

Syntaxe

```
CASE a
        WHEN 1 THEN 'un'
        WHEN 2 THEN 'deux'
        WHEN 3 THEN 'trois'
        ELSE 'autre'
END
```

Dans cet exemple les valeurs contenus dans la colonne “a” sont comparé à 1, 2 ou 3. Si la condition est vrai, alors la valeur située après le THEN sera retournée. la condition ELSE est facultative et sert de ramasse-miette. Si les conditions précédentes ne sont pas respectées alors ce sera la valeur du ELSE qui sera retournée par défaut.

CASE

Exemples

```
SELECT first_name, last_name, points,
  CASE
     WHEN points >= 1000 THEN 'client très fidèle'
     WHEN points < 100  THEN 'client peu fidèle'
     ELSE 'client moyen'
  END
FROM customers;
```

CASE

Exercices

1. Affichez le tableau de nos produits avec une colonne indiquant si le stock est largement suffisant (>50) , normal (<50 et >20) ou critique (<20)
2. Affichez le tableau de nos clients en multipliant leur points par 0.8 si ils sont de Paris, par 0.9 si ils sont de Grenoble ou de Lyon et en ne modifiant pas leur points s'ils viennent d’une autre ville

GROUP BY

La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat.

Syntaxe

```
SELECT colonne1, fonction(colonne2)

FROM table

GROUP BY colonne1
```

Cette commande doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

GROUP BY

Exemples

```
SELECT city, SUM(points)

FROM customers

GROUP BY city
```

```
SELECT city, AVG(points)

FROM customers

GROUP BY city
```

HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX(). Elle permet de sélectionner les colonnes de la table “nom_table” en groupant les lignes qui ont des valeurs identiques sur la colonne “colonne1” et que la condition de HAVING soit respectée.

Syntaxe

```
SELECT colonne1, SUM(colonne2)

FROM nom_table

GROUP BY colonne1

HAVING fonction(colonne2) operateur valeur
```

HAVING

Exemples

```
SELECT city, SUM(points)

FROM customers

GROUP BY city

HAVING SUM(points) > 1000
```

A vous de jouer !

Exercices globaux

1. Affichez le tableau des commandes ayant un nombre d'items > 4
2. Combien de commandes y a t'il en status 1 et 2 ?
3. Combien y a t'il de commandes pour chaque produit du magasin ?
4. Affichez tous les articles ayant un prix supérieur au prix moyen des nos articles de notre magasin.
5. Combien de clients avons nous dans chaque ville ?
6. Affichez les produits en rajoutant une colonne précisant si l'article est en lin ou en coton
7. Affichez le nombre d'items total commandés pour chaque commande, par ordre décroissant.
8. Affichez tous les clients ayant plus de 1000 points par ordre alphabétique
9. Affichez toutes les informations clients en les classant par ville par ordre alphabétique, et pour les clients habitant dans la même ville classez les par leur nombre de points décroissant
10. Quel est le nombre moyen de produits en stock en ne prenant pas en compte les produits ayant un stock <10

A vous de jouer !

Exercices globaux

Grâce au fichiers villes_france.sql et departement.sql, veuillez trouver les requêtes SQL permettant d'effectuer chacune des demandes suivantes :

1. Obtenir la liste des 10 villes les plus peuplées en 2012
2. Obtenir la liste des 50 villes ayant la plus faible superficie
3. Obtenir la liste des départements d'outre-mer, c'est-à-dire ceux dont le numéro de département commencent par "97"
4. Obtenir le nom des 10 villes les plus peuplées en 2012, ainsi que le nom du département associé
5. Obtenir la liste du nom de chaque département, associé à son code et du nombre de commune au sein de ces département, en triant afin d'obtenir en priorité les départements qui possèdent le plus de communes
6. Obtenir la liste des 10 plus grands départements, en terme de superficie
7. Compter le nombre de villes dont le nom commence par "Saint"
8. Obtenir la liste des villes qui ont un nom existants plusieurs fois, et trier afin d'obtenir en premier celles dont le nom est le plus souvent utilisé par plusieurs communes
9. Obtenir en une seule requête SQL la liste des villes dont la superficie est supérieur à la superficie moyenne
10. Obtenir la liste des départements qui possèdent plus de 2 millions d'habitants
11. Remplacez les tirets par un espace vide, pour toutes les villes commençant par "SAINT-" (dans la colonne qui contient les noms en majuscule)