SÉANCE 10

BASES DE DONNÉES : REQUÊTES EN SQL

I - Requêtes dans une table

I.1 - Sélection des attributs (colonnes) et des enregistrements (lignes)

On a vu les premières instructions, les plus fondamentales, au cours précédent :

```
SELECT ... FROM ... WHERE ...
```

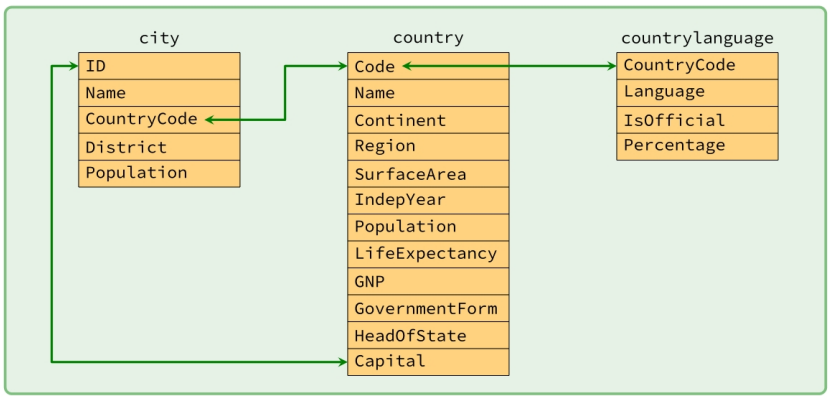
La condition donnée par `WHERE` peut utiliser :

- ▷ les opérateurs de comparaison (=, >, <, <=, >=, <> pour différent) ;
- ▷ les opérateurs logiques `AND`, `OR`, `NOT`.

D'autres opérateurs usuels semblent ne pas être au programme (`IN` qui teste l'appartenance, `LIKE` qui compare des chaînes de caractères d'une certaine forme).

Exemple

Considérons la base de donnée **world** (déjà évoquée au dernier cours) se présentant ainsi :



Pour obtenir la liste des différentes villes de France avec leur population, on réalise la requête :

```
SELECT Name, Population FROM city WHERE CountryCode = 'FRA'
```

Notons que l'on peut **renommer** le nom d'un attribut d'une table à l'aide de `AS` :

```
SELECT a AS b FROM table
```

I.2 - Filtrage des résultats

Tout d'abord, on a vu la façon d'éviter les doublons avec :

```
SELECT DISTINCT ... FROM ...
```

Exemple

Considérons la base de donnée **world**.
Pour obtenir la liste des différentes langues, on réalise la requête : :

```
SELECT DISTINCT Language FROM countrylanguage
```

En fin de requête, on peut utiliser (dans cet ordre) :

- `ORDER BY a DESC` ou `ORDER BY a ASC` pour trier selon l'attribut `a` par ordre décroissant ou croissant (ce dernier étant par défaut) ;
- `LIMIT n` pour limiter la réponse à `n` enregistrements ;
- `OFFSET n` pour débuter la réponse à partir du (`n`+1)-ieme enregistrement.

Exemple / Exercice

Considérons la base de donnée **world**.
Pour obtenir les 5 villes mondiales les plus peuplées, on réalise la requête :

```
SELECT Name FROM city ORDER BY Population DESC LIMIT 5
```

Pour obtenir les 5 suivantes :

```
SELECT Name FROM city ORDER BY Population DESC LIMIT 5 OFFSET 5
```

Pour obtenir le nom des dix pays asiatiques les plus grands :

Pour obtenir le nom et la date d'indépendance des dix pays les plus anciens :

Pour obtenir la liste des langues non officielles pratiquées en France, par ordre décroissant d'importance :

Pour obtenir le nom des cinq pays européens les moins densément peuplés :

II - Opérateurs ensemblistes

On peut appliquer les opérateurs ensemblistes à deux relations `table1` et `table2` différentes.

- ***Union*** : `table1` $\cup$ `table2` avec `UNION`
  L'union de deux ensembles renvoie tous les éléments qui appartiennent à l'un **ou** à l'autre de ces ensembles. Si un élément appartient aux deux à la fois, il n'est renvoyé qu'une seule fois.
- ***Intersection*** : `table1` $\cap$ `table2` avec `INTERSECT`
  L'intersection de deux ensembles renvoie tous les éléments qui appartiennent à l'un **et** à l'autre de ces ensembles.
- ***Différence*** : `table1` $-$ `table2` avec `EXCEPT`
  La différence de deux ensembles A et B envoie les éléments qui appartiennent A mais pas à B.
  Notons que c'est `EXCEPT` qui est utilisé en SQLite mais `MINUS` en MySQL.

Les colonnes des requêtes situées avant et après `INTERSECT` ou `EXCEPT` doivent être similaires (même nombre, même type et même ordre).

Exemple

Considérons deux tables `magasin1` et `magasin2` donnant les clients de deux magasins d'une même enseigne.
Pour obtenir les clients de l'enseigne, on pourra considérer la requête :

```
SELECT * FROM magasin1 UNION magasin2
```

Pour obtenir les clients communs aux deux magasins, on pourra considérer la requête :

```
SELECT * FROM magasin1 INTERSECT magasin2
```

Pour obtenir les clients du `magasin1` qui ne vont pas dans le `magasin2`, on pourra considérer la requête :

```
SELECT * FROM magasin1 EXCEPT magasin2
```

Remarque

Notons que les deux requêtes suivantes sont équivalentes :

```
SELECT a FROM table1 INTERSECT SELECT b FROM table2
```

et :

```
SELECT a FROM table1 WHERE a IN (SELECT b FROM table2)
```

et les deux requêtes suivantes sont équivalentes :

```
SELECT a FROM table1 EXCEPT SELECT b FROM table2
```

et :

```
SELECT a FROM table1 WHERE a NOT IN (SELECT b FROM table2)
```

(entre parenthèses, il s'agit de ***sous-requêtes***).

On peut également considérer le ***produit cartésien*** de deux tables `table1` et `table2` où chaque élément de `table1` est couplé à chaque élément de `table2`.

Exemple

On dispose de deux tables `etudiants` et `villes` :

Nom	Année_de_naissance	Ville
Alice	1997	Toulouse
Bob	1998	Paris
Eve	1995	Toulouse

NomV	Département
Toulouse	31
Paris	75

Le produit cartésien `etudiants`$\times$`villes` fait appel à toutes les colonnes des deux tables :

```
SELECT * FROM etudiants,villes
```

Nom	Année_de_naissance	Ville	NomV	Département
Alice	1997	Toulouse	Toulouse	31
Alice	1997	Toulouse	Paris	75
Bob	1998	Paris	Toulouse	31
Bob	1998	Paris	Paris	75
Eve	1995	Toulouse	Toulouse	31
Eve	1995	Toulouse	Paris	75

Le produit cartésien a peu de sens ici (et en général) car certains étudiants se retrouvent associés à deux villes différentes. Outre ce problème d'interprétation, la taille du résulta pose également problème (produit des cardinaux des deux tables).

III - Jointures

La ***jointure*** est une opération consistant à recoller deux relations `table1` et `table2` en se fondant sur la correspondance de valeurs entre deux colonnes :

```
SELECT * FROM table1 JOIN table2 ON expression_logique
```

Exemple

Reprenons l'exemple précédent. On peut réaliser une jointure selon (`Ville`, `NomV`) entre les tables `etudiants` et `villes`. Cela revient à faire un produit cartésien des deux tables suivi d'une sélection sur les lignes où il y a correspondance entre les deux colonnes.

Nom	Année_de_naissance	Ville	NomV	Département
Alice	1997	Toulouse	Toulouse	31
~~Alice~~	~~1997~~	~~Toulouse~~	~~Paris~~	~~75~~
~~Bob~~	~~1998~~	~~Paris~~	~~Toulouse~~	~~31~~
Bob	1998	Paris	Paris	75
Eve	1995	Toulouse	Toulouse	31
~~Eve~~	~~1995~~	~~Toulouse~~	~~Paris~~	~~75~~

La requête suivante :

```
SELECT * FROM etudiants JOIN villes ON Ville=NomV
```

donne :

Nom	Année_de_naissance	Ville	NomV	Département
Alice	1997	Toulouse	Toulouse	31
Bob	1998	Paris	Paris	75
Eve	1995	Toulouse	Toulouse	31

S'il y a risque d'ambiguïté sur ce que désignent les noms des attributs, on les fait précéder du nom de la table.

Exemple

Dans l'exemple précédent, on peut considérer :

```
SELECT * FROM etudiants JOIN villes ON etudiants.Ville=villes.NomV
```

ou :

```
SELECT * FROM etudiants e JOIN villes v ON e.Ville=v.NomV
```

Notons que l'on peut également ajouter une sélection de colonnes pour éviter d'avoir le champ commun présent deux fois.

Exemple

Dans l'exemple précédent, la requête :

```
SELECT Nom,Année_de_naissance,Ville,Département
  FROM etudiants JOIN villes ON etudiants.Ville=villes.NomV
```

donne :

Nom	Année_de_naissance	Ville	Département
Alice	1997	Toulouse	31
Bob	1998	Paris	75
Eve	1995	Toulouse	31

Exemple / Exercice

Considérons la base de donnée **world**.

Pour obtenir la capitale de l'Ouzbekistan et sa population, on réalise la requête :

```
SELECT city.Name, city.Population
  FROM city JOIN country ON city.ID = country.Capital
 WHERE country.Name = 'Uzbekistan';
```

Pour obtenir la liste des pays ayant adopté le Français comme langue officielle :

Pour obtenir la liste des 5 villes les plus peuplées d'Europe :

IV - Fonctions d'agrégation

Exemple

On dispose de la relation `relevé` ci-dessous à gauche. Un exemple d'agrégation consiste à calculer la moyenne des notes sur chaque classe, ce qui produit la relation ci-dessous à droite.

`relevé`

classe	eleve	note
PC	Meyer	17,5
PSI	Martin	7,75
PC	Bernard	9,25
PSI	Robert	14,0
PSI	Dubois	11,5

classe	moyenne
PC	13,38
PSI	11,08

L'agrégation sert à regrouper les lignes d'une table et à évaluer une fonction f sur ces regroupements.
Les cinq fonctions à connaître sont :

Fonction	Syntaxe SQL	Valeur renvoyée
comptage	`COUNT`	nombre d'éléments
moyenne	`AVG`	moyenne des éléments
somme	`SUM`	somme des éléments
min	`MIN`	le plus petit des éléments
max	`MAX`	le plus grand des éléments

Exemple

Considérons la table `utilisateur` donnant la liste des utilisateurs d'un site web :

id	nom	ville	date_inscription	nombre_achat	id_dernier_achat
1	Marie	Paris	2010-04-22	5	24
2	Léon	Marseille	2011-08-18	3	36
3	Paul	Lyon	2011-11-02	0	NULL
4	Léon	Paris	2012-09-01	1	7
5	Paul	Nantes	2013-01-10	0	NULL

Pour compter le nombre d'utilisateurs qui ont effectué au moins un achat :

```
SELECT COUNT(*) AS nb_user FROM utilisateur WHERE nombre_achat>0
```

nb_user
3

Pour calculer le nombre total d'achats effectués sur le site par les personnes se nommant Paul :

```
SELECT SUM(nombre_achat) AS achat_tot FROM utilisateur WHERE nom='Paul'
```

achat_tot
0

Il est aussi possible de regrouper les enregistrements d'une table par agrégation à l'aide du mot-clef `GROUP BY`. Ce regroupement permet d'appliquer la fonction à chacun des groupes et le résultat de la requête est l'ensemble des valeurs prises par la fonction sur chacun des regroupements.
La syntaxe est (où `a` et `b` sont des attributs et `f` est la fonction d'agrégation) :

```
SELECT f(a) FROM table WHERE expression_logique GROUP BY b
```

Exemple
Poursuivons le même exemple que ci-dessus. Pour calculer le nombre d'achats effectués par chaque nom, on regroupe les lignes :

```
SELECT nom, SUM(nombre_achat) AS achat_tot FROM utilisateur GROUP BY nom
```

nom	achat_tot
Marie	5
Léon	4
Paul	0

Exemple
Reprenons l'exemple initial de ce paragraphe : une requête permettant de calculer les moyennes des notes sur chaque classe s'écrit :

```
SELECT classe, AVG(note) AS moyenne FROM relevé GROUP BY classe
```

classe	moyenne
PC	13,38
PSI	11,08

Lorsque l'on impose une contrainte (par exemple une sélection) **après** avoir utilisé une fonction d'agrégation, en particulier après `GROUP BY`, il faut utiliser le mot-clé `HAVING` et non `WHERE`.

Exemple
Considérons la base de donnée **world**.
Pour obtenir la population de chacun des continents (dans l'ordre décroissant), on réalise la requête :

```
SELECT Continent, SUM(Population) AS Pop FROM country GROUP BY Continent ORDER BY Pop DESC
```

Si l'on souhaite maintenant se restreindre aux continents comportant plus de 40 pays, on écrit :

```
SELECT Continent, SUM(Population), COUNT(*) AS NombreDePays FROM country
GROUP BY Continent HAVING NombreDePays > 40
```

V - Des exercices issus de sujets de concours

V.1 - MP-PC-PSI Mines-Ponts 2017

On modélise un réseau routier par un ensemble de croisements et de voies reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés. La base de données du réseau routier est constituée des relations suivantes :

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère *c* l'identifiant (id) d'un croisement donné.

Question 1 - Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.

Question 2 - Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c.

Question 3 - Que renvoie la requête SQL suivante?

```sql
SELECT V2.id_croisement_fin
  FROM Voie as V1 JOIN Voie as V2
    ON V1.id_croisement_fin = V2.id_croisement_debut
  WHERE V1.id_croisement_debut = c
```

V.2 - MP-PC-PSI Centrale-Supélec 2016

Ce problème s'intéresse à différents aspects relatifs à la sécurité aérienne et plus précisément au risque de collision entre deux appareils.
Afin d'éviter les collisions entre avions, les altitudes de vol en croisière sont normalisées. Dans la majorité des pays, les avions volent à une altitude multiple de 1000 pieds (un pied vaut 30,48 cm) au-dessus de la surface isobare à 1013,25 hPa. L'espace aérien est ainsi découpé en tranches horizontales appelées niveaux de vol et désignées par les lettres « FL » (*flight level*) suivies de l'altitude en centaines de pieds : « FL310 » désigne une altitude de croisière de 31000 pieds au-dessus de la surface isobare de référence.

EUROCONTROL est l'organisation européenne chargée de la navigation aérienne, elle gère plusieurs dizaines de milliers de vol par jour. Toute compagnie qui souhaite faire traverser le ciel européen à un de ses avions doit soumettre à cet organisme un plan de vol comprenant un certain nombre d'informations : trajet, heure de départ, niveau de vol souhaité, etc. Muni de ces informations, Eurocontrol peut prévoir les secteurs aériens qui vont être surchargés et prendre des mesures en conséquence pour les désengorger : retard au décollage, modification de la route à suivre, etc.

Nous modélisons (de manière très simplifiée) les plans de vol gérés par EUROCONTROL sous la forme d'une base de données comportant deux tables :

– la table `vol` qui répertorie les plans de vol déposés par les compagnies aériennes ; elle contient les colonnes
  - `id_vol` : numéro du vol (chaine de caractères) ;
  - `depart` : code de l'aéroport de départ (chaine de caractères) ;
  - `arrivee` : code de l'aéroport d'arrivée (chaine de caractères) ;
  - `jour` : jour du vol (de type date, affiché au format `aaaa-mm-jj`) ;
  - `heure` : heure de décollage souhaitée (de type time, affiché au format `hh :mi`) ;
  - `niveau` : niveau de vol souhaité (entier).

id_vol	depart	arrivee	jour	heure	niveau
AF1204	CDG	FCO	2016-05-02	07 :35	300
AF1205	FCO	CDG	2016-05-02	10 :25	300
AF1504	CDG	FCO	2016-05-02	10 :05	310
AF1505	FCO	CDG	2016-05-02	13 :00	310

Figure 1 Extrait de la table `vol` : vols de la compagnie Air France entre les aéroports Charles-de-Gaule (Paris) et Léonard-de-Vinci à Fiumicino (Rome)

– la table `aeroport` qui répertorie les aéroports européens ; elle contient les colonnes
  - `id_aero` : code de l'aéroport (chaine de caractères) ;
  - `ville` : principale ville desservie (chaine de caractères) ;

- `pays` : pays dans lequel se situe l'aéroport (chaine de caractères).

id_aero	ville	pays
CDG	Paris	France
ORY	Paris	France
MRS	Marseille	France
FCO	Rome	Italie

Figure 2 Extrait de la table `aeroport`

Les types SQL `date` et `time` permettent de mémoriser respectivement un jour du calendrier grégorien et une heure du jour. Deux valeurs de type `date` ou de type `time` peuvent être comparées avec les opérateurs habituels(=, <, <=, etc.). La comparaison s'effectue suivant l'ordre chronologique. Ces valeurs peuvent également être comparées à une chaine de caractères correspondant à leur représentation externe (`'aaaa-mm-jj'` ou `'hh :mi'`).

Question 1 - Écrire une requête SQL qui fournit le nombre de vols qui doivent décoller dans la journée du 2 mai 2016 avant midi.

Question 2 - Écrire une requête SQL qui fournit la liste des numéros de vols au départ d'un aéroport desservant Paris le 2 mai 2016

Question 3 - Que fait la requête suivante ?

```
SELECT id_vol
  FROM vol  JOIN aeroport AS d ON d.id_aero = depart
            JOIN aeroport AS a ON a.id_aero = arrivee
  WHERE
     d.pays = 'France' AND
     a.pays = 'France' AND
     jour = '2016-05-02'
```

Question 4 - Certains vols peuvent engendrer des conflits potentiels : c'est par exemple le cas lorsque deux avions suivent un même trajet, en sens inverse, le même jour et à un même niveau. Écrire une requête SQL qui fournit la liste des couples (Id_1 , Id_2) des identifiants des vols dans cette situation.

V.3 - PC Mines Ponts 2019

Chiffrer les données est nécessaire pour assurer la confidentialité lors d'échanges d'informations sensibles. Dans ce domaine, les nombres premiers servent de base au principe de clés publique et privée qui permettent, au travers d'algorithmes, d'échanger des messages chiffrés. La sécurité de cette méthode de chiffrement repose sur l'existence d'opérations mathématiques peu coûteuses en temps d'exécution mais dont l'inversion (c'est-à-dire la détermination des opérandes de départ à partir du résultat) prend un temps exorbitant. On appelle ces opérations "fonctions à sens unique". Une telle opération est, par exemple, la multiplication de grands nombres premiers. Il est aisé de calculer leur produit. Par contre, connaissant uniquement ce produit, il est très difficile de déduire les deux facteurs premiers.

[...]

Les questions portant sur les bases de données sont à traiter en langage SQL.

Au cours du développement des fonctions nécessaires à la manipulation des nombres premiers on s'aperçoit que le choix des algorithmes pour évaluer chaque fonction est primordial pour garantir des performances acceptables. On souhaite donc mener des tests à grande échelle pour évaluer les performances réelles du code qui a été développé. Pour ce faire on effectue un grand nombre de tests sur une multitude d'ordinateurs. Les données sont ensuite centralisées dans une base de données composée de deux tables.

La première table est `ordinateurs` et permet de stocker des informations sur les ordinateurs utilisés pour les tests. Ses attributs sont :

- `nom TEXT`, clé primaire, le nom de l'ordinateur.
- `gflops INTEGER` la puissance de l'ordinateur en milliards d'opérations flottantes par seconde.
- `ram INTEGER` la quantité de mémoire vive de l'ordinateur en Go.

Exemple du contenu de cette table :

nom	gflops	ram
nyarlathotep114	69	32
nyarlathotep119	137	32
...		
shubniggurath42	133	16
azathoth137	85	8

La seconde table est `fonctions` et stocke les informations sur les tests effectués pour différentes fonctions en cours de développement. Ses attributs sont :

- `id INTEGER` l'identifiant du test effectué.
- `nom TEXT` le nom de la fonction testée (par exemple li, Ei, etc).
- `algorithme TEXT` le nom de l'algorithme qui permet le calcul de la fonction testée (par exemple BBS si on teste une fonction de génération de nombres aléatoires).
- `teste_sur TEXT` le nom du PC sur lequel le test a été effectué.
- `temps_exec INTEGER` le temps d'exécution du test en millisecondes.

Exemple du contenu de cette table :

id	nom	algorithme	teste_sur	temps_exec
1	li	rectangles	nyarlathotep165	2638
2	li	rectangles	shubniggurath28	736
3	li	trapezes	nyarlathotep165	4842
...				
2154	Ei	puiseux	nyarlathotep145	2766
2155	aleatoire	BBS	azathoth145	524

Question 1 Expliquer pourquoi il n'est pas possible d'utiliser l'attribut `nom` comme clé primaire de la table `fonctions`.

Question 2 Écrire des requêtes SQL permettant de :

1. Connaître le nombre d'ordinateurs disponibles et leur quantité moyenne de mémoire vive.
2. Extraire les noms des PC sur lesquels l'algorithme `rectangles` n'a pas été testé pour la fonction nommée `li`.
3. Pour la fonction nommée `Ei`, trier les résultats des tests du plus lent au plus rapide. Pour chaque test retenir le nom de l'algorithme utilisé, le nom du pc sur lequel il a été effectué et la puissance du PC.

V.4 - PC X 2019

On souhaite utiliser une base de données pour stocker les résultats obtenus par une communauté de joueurs. On suppose que l'on dispose d'une base de données comportant les tables `JOUEURS(id_j, nom, pays)` et `PARTIES(id_p, date, duree, score, id_joueur)` où :

- `id_j`, de type entier, est la clé primaire de la table `JOUEURS`,
- `nom` est une chaîne de caractères donnant le nom du joueur,
- `pays` est une chaîne de caractères donnant le pays du joueur,
- `id_p`, de type entier, est la clé primaire de la table `PARTIES`,
- `date` est la date (`AAAAMMJJ`) de la partie,
- `duree`, de type entier, est la durée en secondes de la partie,
- `score`, de type entier, est le nombre de points marqués au cours de la partie,
- `id_joueur` est un entier qui identifie le joueur de la partie.

Question 1 Étant donné une chaîne de caractères `cc` contenant le nom d'un joueur, écrire une requête `SQL` qui renvoie la date, la durée et le score de toutes les parties jouées par le joueur `cc`, listées par ordre chronologique (au choix, croissant ou décroissant).

Question 2 Étant donné un entier s (le score que vient de réaliser une joueuse nommée Alice), écrire une requête `SQL` qui renvoie la position qu'aura le score s dans le classement des parties par ordre de score (on suppose que la dernière partie d'Alice n'a pas encore été insérée dans la table des parties). En cas d'ex æquo pour le score s (une ou plusieurs parties déjà présentes ayant le score s), le rang sera le même que s'il n'y avait qu'une seule partie avec le score s.
Par exemple, la requête renverra 1 (le score d'Alice est "1er") si aucun score n'est meilleur que s. Autre exemple, si la base de données contient 6 parties dont les scores sont 87, 75, 75, 63, 60, 60, alors le rang de $s = 75$ sera 2, le rang de $s = 70$ sera 4 et le rang de $s = 60$ sera 5.

Question 3 Écrire une requête `SQL` qui renvoie le record de France de Tetris couleur, c'est-à-dire le meilleur score réalisé par un joueur dont le pays est la France.

Question 4 Étant donné une chaîne de caractères `cc` contenant le nom d'un joueur (ayant déjà joué au moins une partie de Tetris couleur), écrire une requête `SQL` qui renvoie le rang du joueur `cc`, c'est-à-dire sa position dans le classement des joueurs par ordre de leur meilleur score dans une partie de Tetris couleur (on traitera les ex æquo de la même manière qu'à la question 4).

VI - Éléments de correction

Page 2 –

```sql
SELECT Name, SurfaceArea
  FROM country
 WHERE Continent = "Asia"
 ORDER BY SurfaceArea DESC
 LIMIT 10
```

```sql
SELECT Name, IndepYear
  FROM country
 WHERE IndepYear > -5000
 ORDER BY IndepYear ASC
 LIMIT 10
```

```sql
SELECT Language, Percentage
  FROM countrylanguage
 WHERE CountryCode = "FRA" AND IsOfficial = "F"
 ORDER BY Percentage DESC
```

```sql
SELECT Name, Population / SurfaceArea AS densite
  FROM country
 WHERE Continent = "Europe"
 ORDER BY densite ASC
 LIMIT 5
```

ou plus simplement :

```sql
SELECT Name
  FROM country
 WHERE Continent = "Europe"
 ORDER BY Population / SurfaceArea ASC
 LIMIT 5
```

Page 5 –

```sql
SELECT country.Name
  FROM countrylanguage
       JOIN
       country ON countrylanguage.CountryCode = country.Code
 WHERE countrylanguage.Language = 'French' AND
       IsOfficial = 'T'

SELECT city.Name,
       city.Population
  FROM city
       JOIN
       country ON city.CountryCode = country.Code
 WHERE country.Continent = 'Europe'
 ORDER BY city.Population DESC
 LIMIT 5
```

VI.1 - MP-PC-PSI Mines-Ponts 2017

Question 1 -

```
SELECT id_croisement_fin FROM voie WHERE id_croisement_debut = c;
```

Question 2 -

```
SELECT longitude, latitude FROM croisement cr
  JOIN voie ON cr.id = id_croisement_fin
    WHERE id_croisement_debut = c;
```

Question 3 -
Cette requête renvoie les identifiants des croisements atteignables en utilisant deux voies à partir du croisement ayant l'identifiant c (donc avec un croisement intermédiaire).

VI.2 - MP-PC-PSI Centrale-Supélec 2016

Question 1 -
Dans l'hypothèse où midi est inclus (sinon en remplace <= par <) :

```
SELECT COUNT(*) FROM vol WHERE jour = '2016-05-02' AND heure <= '12:00'
```

Question 2 -
Trois interprétations possibles :

– Numéro des vols (v1) tels qu'il existe un vol (v2) partant du même aéroport et arrivant dans l'un des aéroports de Paris le 2 mai 2016 :

```
SELECT v1.id_vol FROM vol v1 JOIN vol v2   ON v1.depart = v2.depart
                             JOIN aeroport ON v2.arrivee = id_aero
  WHERE ville = 'Paris' AND v2.jour = '2016-05-02'
```

– Numéro des vols arrivant dans l'un des aéroports de Paris le 2 mai 2016 :

```
SELECT id_vol FROM vol JOIN aeroport ON arrivee = id_aero
  WHERE ville = 'Paris' AND jour = '2016-05-02'
```

– Numéro des vols partant de l'un des aéroports de Paris le 2 mai 2016 :

```
SELECT id_vol FROM vol JOIN aeroport ON depart = id_aero
  WHERE ville = 'Paris' AND jour = '2016-05-02'
```

Question 3 -
Cette requête renvoie les numéros de vols entre deux villes françaises le 2 mai 2016.

Question 4 -

```
SELECT v1.id_vol, v2.id_vol FROM vol v1
  JOIN vol v2 ON v1.niveau  = v2.niveau
             AND v1.depart  = v2.arrivee
             AND v1.arrivee = v2.depart
             AND v1.jour    = v2.jour
  WHERE v1.id_vol < v2.id_vol
```

VI.3 - PC Mines Ponts 2019

Question 1 Plusieurs enregistrements ont la même valeur pour le champ `nom` car la même fonction peut être testée plusieurs fois, sur différents ordinateurs par exemple ou pour différents algorithmes.

Question 2

1.
```sql
SELECT COUNT(*), AVG(ram) FROM ordinateurs
```

2.
```sql
SELECT nom FROM ordinateurs
    WHERE nom NOT IN (
    SELECT teste_sur FROM fonctions
        WHERE nom = "li" AND algorithme = "rectangles"
    )
```

ou

```sql
SELECT nom FROM ordinateurs
EXCEPT
SELECT teste_sur FROM fonctions
    WHERE nom = "li" AND algorithme = "rectangles"
```

3.
```sql
SELECT algorithme, teste_sur, gflops FROM fonctions JOIN ordinateurs
ON fonctions.teste_sur = ordinateurs.nom
WHERE fonctions.nom = "Ei"
ORDER BY temps_exec DESC
```

VI.4 - PC X 2019

Question 1

```sql
SELECT date, duree, score
FROM PARTIES JOIN JOUEURS ON id_joueur = id_j
WHERE nom = cc
ORDER BY date
```

Question 2

```sql
SELECT COUNT(*) + 1 FROM PARTIES WHERE score > s
```

Question 3

```sql
SELECT MAX(score)
FROM PARTIES JOIN JOUEURS ON id_joueur = id_j
WHERE pays = "France"
```

Question 4

```sql
SELECT COUNT(*) + 1
FROM (
SELECT id_j FROM PARTIES
GROUP BY id_j HAVING MAX(score) > ( SELECT MAX(score)
                                    FROM PARTIES JOIN JOUEURS
                                    ON id_joueur = id_j
                                    WHERE nom = cc
                                   )
    )
```