Chapitre 4

Sélection de données en langage SQL

Nous avons découvert dans les cours précédents la notion de **base de données** et avons effectué nos premières opérations d'insertion et de recherche via les outils graphiques de PhpMyAdmin. Aujourd'hui, nous allons découvrir ce qui se cache derrière ces outils, à savoir le **langage SQL**.

En effet, si l'on souhaite accéder et échanger avec notre base de données depuis un site WEB ou un script Python, PhpMyAdmin ne nous sera d'aucun secours. Il faudra « dialoguer » avec notre base de données dans un langage qu'elle puisse interpréter, le SQL. Dialoguer avec notre base, c'est lui soumettre des **requêtes SQL**.

Dans ce TP, nous importerons la base de données suivantes dans phpMyAdmin :

`http://nsi.dellasantina.corsica/documents/BDD/NSI.sql`

Cette base de données contient les 3 tables *utilisateurs*, *loisirs* et *communes* déjà utilisées dans le chapitre 2.

L'emploi du langage SQL se fera à travers l'éditeur SQL de PhpMyAdmin. Nous pourrons y tester toutes nos requêtes et voir les éventuels problèmes qui pourraient survenir.

4.1 Requêtes de sélection

4.1.1 La requête `SELECT`

La requête qui permet de sélectionner et d'afficher des données s'appelle `SELECT`.
On précise la/les colonne(s) à sélectionner ainsi que la table utilisée avec le mot-clé `FROM` de la manière suivante :

```
SELECT colonne1, colonne2, ...
FROM nom_table
```

Par exemple, pour sélectionner les noms et prénoms des utilisateurs de la table *utilisateurs*, on écrira :

```
SELECT nom, prenom
FROM utilisateurs
```

Il est également possible de sélectionner toutes les colonnes d'une table avec le caractère `*`.

```
SELECT *
FROM utilisateurs
```

Il est possible de renommer certaines colonnes dans la sélection : la table ne sera pas modifiée, mais le nom des colonnes apparaîtra différent lors de l'affichage. On utilise pour cela le mot-clé `AS`. Nous verrons plus loin des exemples d'utilisations concrets.

```
SELECT nom, prenom
FROM utilisateurs
```

nom	prenom
Orsoni	Françoise
Filippi	Sophie
Mondoloni	Andria
Mattei	Julien
Marcaggi	Albert
Orsini	Paolo
Tasso	Saveriu
Cesari	Sophie

```
SELECT nom AS nom_de_famille, prenom
FROM utilisateurs
```

nom_de_famille	prenom
Orsoni	Françoise
Filippi	Sophie
Mondoloni	Andria
Mattei	Julien
Marcaggi	Albert
Orsini	Paolo
Tasso	Saveriu
Cesari	Sophie

Remarque. Les mots-clés SQL comme `SELECT` et `FROM` sont écrits en majuscules par convention, mais MySQL n'est pas sensible à la casse. Attention cependant, ce n'est pas le cas pour le nom des tables !

Remarque. Une requête SQL peut s'écrire sur une seule ligne. Les sauts de lignes permettent seulement plus de lisibilité.

Question 01 Écrire une requête SQL permettant de sélectionner l'id et l'âge de chaque utilisateur de la table *utilisateurs*.

Question 02 Écrire une requête SQL permettant de sélectionner le nom et la population de chaque commune de la table *communes*.

4.1.2 La clause `WHERE`

La clause `WHERE` permet de restreindre la sélection selon certains critères :

```
SELECT colonne1, colonne2, ...
FROM nom_table
WHERE criteres
```

Un critère de recherche peut être une égalité, une inégalité, une différence... ou une combinaison de tout cela.
Par exemple, pour obtenir les noms et prénoms des utilisateurs ayant exactement 40 ans :

```
SELECT nom, prenom
FROM utilisateurs
WHERE age = 40
```

Lorsque la colonne est de type chaîne de caractères, on entoure la valeur cherchée par des guillemets :

```
SELECT nom, prenom
FROM utilisateurs
WHERE nom = 'CASANOVA' -- Utilisateurs dont le nom de famille est CASANOVA
```

Quelques exemples

Identifiants des utilisateurs de moins de 50 ans (inclus)

```
SELECT id
FROM utilisateurs
WHERE age <= 50
```

Âges des utilisateurs dont le prénom est Sophie :

```
SELECT age
FROM utilisateurs
WHERE prenom = 'Sophie'
```

Question 03 Écrire une requête SQL permettant de sélectionner :

1. le prénom de tous les utilisateurs âgés d'au moins 30 ans
2. le nom et le code INSEE des communes de Corse-du-Sud

Question 04 Décrire les requêtes SQL suivantes :

```
SELECT nom_commune
FROM communes
WHERE population < 2000
```

```
SELECT id_utilisateur
FROM loisirs
WHERE intitule = 'Cuisine'
```

Il est possible de combiner les critères avec les opérateurs logiques `OR`, `AND` et `NOT`.

Quelques exemples

Emails des utilisateurs âgés de moins de 40 ans et dont le nom de famille est *Mattei*

```
SELECT email
FROM utilisateurs
WHERE nom = 'Mattei' AND age <= 40
```

Identifiants et âges des utilisateurs dont le prénom est Sophie ou Jean :

```
SELECT id, age
FROM utilisateurs
WHERE prenom = 'Sophie' OR prenom = 'Jean'
```

Le mot-clé `BETWEEN` permet de spécifier un encadrement entre 2 valeurs :

```
-- Sélection des noms de communes comprenant entre 2000 et 5000 habitants
SELECT nom_commune
FROM communes
WHERE population BETWEEN 2000 AND 5000
```

Le mot-clé `IN` permet de spécifier un ensemble discret de valeurs :

```
-- Sélection des utilisateurs dont le nom de famille est Casanova, Mattei ou Renucci
SELECT *
FROM utilisateurs
WHERE nom IN ('Mattei', 'Casanova', 'Renucci')
```

Le mot-clé `LIKE` permet de comparer des chaînes de caractères :

```
-- Sélection des utilisateurs dont le nom de famille commence par M
SELECT *
FROM utilisateurs
WHERE nom LIKE 'M%'
```

Question 05 Écrire une requête SQL permettant de sélectionner :

1. les identifiants utilisateurs dont le loisir est la danse ou le rallye
2. les communes de Haute-Corse de plus de 5000 habitants
3. les utilisateurs dont les initiales sont AC (comme Andria Casanova)

4.1.3 Trier les données avec `ORDER BY`

Lors d'un `SELECT` , Mysql nous retourne les données dans un certain ordre, qui n'est pas toujours clair. Pour préciser l'ordre de sélection, on utilise le mot-clé `ORDER BY` (après les critères de sélection de `WHERE`) en précisant sur quelle(s) colonne(s) effectuer le tri (dans le sens croissant par défaut).

```
-- Sélection des clients par âge croissant
SELECT *
FROM utilisateurs
ORDER BY age

-- Sélection des communes de Corse-du-Sud dans l'ordre alphabétique
SELECT *
FROM communes
WHERE code_departement = '2A'
ORDER BY nom_commune
```

Spécifier plusieurs colonnes permet d'effectuer le tri sur chaque colonne, dans l'ordre donné. Pratique lorsque une valeur peut se répéter plusieurs fois, comme un nom de famille.

```
-- Sélection des utilisateurs l'ordre alphabétique (nom-prénom)
SELECT *
FROM utilisateurs
ORDER BY nom, prenom
```

Le mot-clé `ASC` permet de préciser que le tri est croissant (**asc**endant), c'est la valeur par défaut. Le mot-clé `DESC` permet d'effectuer un tri décroissant (**desc**endant).

```
-- Sélection des communes dans l'ordre décroissant de population
SELECT *
FROM communes
ORDER BY population DESC
```

Question 06 Écrire une requête SQL permettant de sélectionner :

1. les communes de moins de 4000 habitants, dans l'ordre alphabétique
2. les utilisateurs de moins de 30 ans, du plus vieux au plus jeune, et dans l'ordre alphabétique (âge en premier)

4.1.4 Limiter le nombre de résultats avec `LIMIT`

Souvent, il n'est pas nécessaire d'afficher toutes les lignes d'une table mais seulement une partie. On peut faire cela grâce au mot-clé `LIMIT`, qui s'utilise après le mot-clé `ORDER BY` (s'il est utilisé) et donc nécessairement après la clause `WHERE`.
`LIMIT` s'utilise avec 2 paramètres :

- le nombre de lignes à récupérer
- un décalage indiquant à partir de quelle ligne on récupère les résultats (par défaut égal à 0)

```
LIMIT nombre_lignes                -- Pas de décalage précisé
LIMIT nombre_lignes OFFSET decalage -- Décalage précisé
LIMIT decalage, nombre_lignes       -- Autre façon d'écrire avec le décalage
```

Par exemple, pour sélectionner les 5 premiers utilisateurs triés par ordre alphabétique :

```
SELECT *
FROM utilisateurs
ORDER BY nom, prenom
LIMIT 5
```

On peut limiter le nombre de résultats sans forcément trier ces derniers :

```
-- Sélection de 20 communes de Haute-Corse
SELECT *
FROM communes
WHERE code_departement = '2B'
LIMIT 20
```

Question 07 Écrire une requête SQL permettant de sélectionner :

1. le nom des 5 communes les moins peuplées de Corse
2. 10 utilisateurs âgés de plus de 30 ans

4.1.5 Éliminer les doublons avec `DISTINCT`

Enfin, il est possible d'éliminer les doublons dans une sélection avec le mot-clé `DISTINCT` , lors du choix des colonnes :

```
SELECT DISTINCT colonne1
FROM nom_table
```

```
-- Sélection des noms de famille présents dans la table utilisateurs, sans doublons
SELECT DISTINCT nom
FROM utilisateurs

-- Sélection des noms de famille des utilisateurs de moins de 30 ans, sans doublons
SELECT DISTINCT nom
FROM utilisateurs
WHERE age <= 30
```

On peut préciser plusieurs colonnes avec `DISTINCT` : dans ce cas, on ne sélectionnera que les lignes dont TOUTES les colonnes sont distinctes.

```
-- Sélection des couples noms de famille / prénoms distincts
SELECT DISTINCT nom, prenom
FROM utilisateurs
```

`DISTINCT` est très pratique, notamment dans les tables où il y a beaucoup de doublons, comme dans notre table *loisirs*, où plusieurs utilisateurs peuvent avoir le même loisir.

Question 08 Écrire une requête SQL permettant de sélectionner :

1. les différents prénoms de la table *utilisateurs*, sans doublons
2. l'intitulé des loisirs de la table *loisirs*, sans doublons, dans l'ordre alphabétique
3. les 3 plus grands âges distincts de la table *utilisateurs*
4. le nombre de communes possédant au moins un utilisateur

4.2 Fonctions d'agréation

Les fonctions d'agrégation sont des fonctions qui vont regrouper les lignes. Elles agissent sur une colonne et renvoient un résultat unique pour toutes les lignes sélectionnées. Elles sont utilisées pour compter et faire des statistiques sur les données.

4.2.1 Compter les lignes avec `COUNT`

`COUNT` est la fonction d'agréation la plus utilisée. Elle permet de compter les lignes d'une sélection.

```sql
-- Nombre de lignes de la table utilisateurs
SELECT COUNT(*)
FROM utilisateurs
```

On précise toujours une colonne sur laquelle effectuer le comptage. Généralement, on sélectionne toutes les colonnes et on écrit `COUNT(*)` mais on aurait pu très bien écrire `COUNT(nom)` . Néanmoins, il y a une différence : `COUNT(colonne)` ne comptera pas les éventuelles valeurs `NULL` .

```sql
-- Nombre de lignes
SELECT COUNT(*)
FROM utilisateurs
```

```sql
-- Nombre de lignes où nom != NULL
SELECT COUNT(nom)
FROM utilisateurs
```

Grâce au mot-clé `AS` que nous avons vu en début de chapitre, il est possible de nommer le nombre de lignes ainsi calculé.

```sql
SELECT COUNT(*) AS nombre_utilisateurs
FROM utilisateurs
```

Et bien sûr, on peut combiner `COUNT` avec les critères de sélection et de tri vus précédemment.

```sql
-- Nombre d'utilisateurs ayant exactement 40 ans
SELECT COUNT(*)
FROM utilisateurs
WHERE age = 40
```

On peut même combiner `COUNT` et `DISTINCT` :

```sql
-- Nombre de noms de famille distincts
SELECT COUNT(DISTINCT nom)
FROM utilisateurs
```

Question 09 Écrire une requête SQL permettant de sélectionner :

1. le nombre d'utilisateurs prénommés Jean-Paul
2. le nombre de loisirs distincts dans la table *loisirs*
3. le nombre de communes de plus de 200 habitants commençant par un A

4.2.2 Minimum et maximum avec `MIN` et `MAX`

Comme leurs noms l'indique, `MIN` et `MAX` permettent de sélectionner le minimum ou le maximum d'une colonne.

```
-- Âges minimum et maximum dans la table utilisateurs
SELECT MIN(age) AS age_minimum, MAX(age) AS age_maximum
FROM utilisateurs
```

On fera tout de même **attention aux requêtes mélangeant fonctions d'agrégats et sélection normale**, comme la requête suivante (dont on peut s'interroger sur le sens profond) :

```
-- Sélection de la plus petite latitude et du nom
SELECT MIN(latitude), nom_commune
FROM communes
```

Contrairement à ce que l'on pourrait penser, le nom sélectionné ne correspond pas à la plus petite latitude sélectionnée ! Ici, l'utilisation d'une fonction d'agrégat fait que le résultat obtenu tient sur **une seule ligne**. Ainsi, MySQL va sélectionner un nom au hasard (généralement celui de la première ligne des résultats obtenus sans utilisation de la fonction d'agrégation).

Question 10 Proposer une requête SQL permettant de sélectionner le nom de la commune la plus au Sud de Corse.

4.2.3 Somme et moyenne avec `SUM` et `AVG`

`SUM` permet de calculer la **somme** sur la colonne donnée.

```
-- Population totale en Corse
SELECT SUM(population) AS population_totale
FROM communes
```

`AVG` permet de calculer la **moyenne** sur la colonne donnée.

```
-- Âge moyen des utilisateurs
SELECT AVG(age) AS age_moyen
FROM utilisateurs
```

Question 11 Écrire une requête SQL permettant de sélectionner :

1. l'âge moyen des utilisateurs prénomés Sophie
2. la population totale en Corse-du-Sud

4.3 Jointures

Les tables sur lesquelles nous travaillons depuis le début de ce TP possèdent des relations internes :

- les colonnes *code_commune* de la table *utilisateurs* et de la table *communes*
- les colonnes *id* de la table *utilisateurs* et *id_utilisateur* de la table *loisirs*

Ces colonnes, partageant des valeurs communes, permettent de faire le lien entre deux tables. L'intérêt principal est d'**éviter la redondance des données** mais aussi de **séparer les informations**.

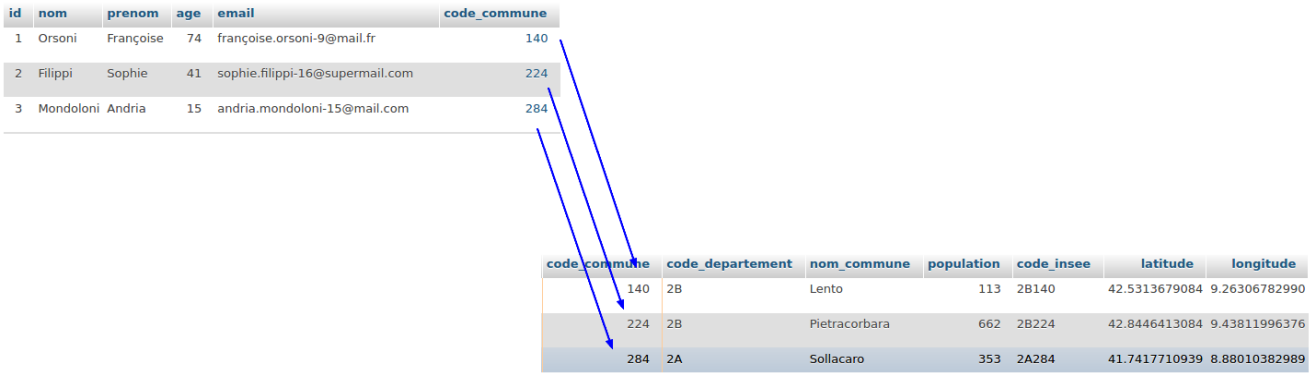
On aurait pu utiliser une table *utilisateurs* contenant - en plus des informations sur l'utilisateur comme son nom, son email et son âge - le nom de la commune de l'utilisateur ainsi que toutes les informations s'y rattachant, comme les coordonnées géographiques ou la population. Mais cela pose plusieurs problèmes :

- Si 100 utilisateurs habitent à Ajaccio, les informations sur la ville d'Ajaccio seraient répétées 100 fois (population, coordonnées, code INSEE...)
- Une même ligne contiendrait un email et une population, alors que ces informations n'ont rien à voir

On préfère ainsi créer plusieurs tables, contenant des informations similaires :

- Une table *utilisateurs*, contenant les informations propres chaque l'utilisateur
- Une table *communes*, contenant les informations propres à chaque commune

Chaque commune est identifiée par un nombre unique, le **code_commune**, et chaque utilisateur possède un code commune, permettant d'identifier la commune qu'il habite.



id	nom	prenom	age	email	code_commune
1	Orsoni	Françoise	74	françoise.orsoni-9@mail.fr	140
2	Filippi	Sophie	41	sophie.filippi-16@supermail.com	224
3	Mondoloni	Andria	15	andria.mondoloni-15@mail.com	284

code_commune	code_departement	nom_commune	population	code_insee	latitude	longitude
140	2B	Lento	113	2B140	42.5313679084	9.26306782990
224	2B	Pietracorbara	662	2B224	42.8446413084	9.43811996376
284	2A	Sollacaro	353	2A284	41.7417710939	8.88010382989

Problème. *Comment savoir le nom de la commune dans laquelle habite un utilisateur donné ?*

C'est là qu'interviennent les **jointures**.

Pour réaliser la jointure précédente - appelée jointure **interne** - on utilise la syntaxe suivante :

```
SELECT *
FROM utilisateurs
JOIN communes
  ON communes.code_commune = utilisateurs.code_commune
```

Examinons ligne par ligne la requête précédente :

- On sélectionne toutes les colonnes :

```
SELECT *
```

- Depuis la table *utilisateurs* :

```
FROM utilisateurs
```

- En joignant la table *communes* :

```
JOIN communes
```

- Et en précisant **sur quelle colonne se fait la jointure** :

```
  ON communes.code_commune = utilisateurs.code_commune
```

La dernière ligne est un peu particulière : on serait tentés d'écrire `ON code_commune` mais le nom de la colonne jointe peut différer d'une table à l'autre (c'est d'ailleurs le cas pour la table *loisirs* comme nous le verrons plus tard).
On précise à quelle table appartient la colonne considérée en écrivant `nom_table.colonne` : c'est d'ailleurs bien utile pour distinguer chacune de ces colonnes (même si elles ont dans ce cas les mêmes valeurs...).

id	nom	prenom	age	email	code_commune	code_commune	code_departement	nom_commune	population	code_insee	latitude	longitude
1	Orsoni	Françoise	74	françoise.orsoni-9@mail.fr	140	140	2B	Lento	113	2B140	42.5313679084	9.26306782990
2	Filippi	Sophie	41	sophie.filippi-16@supermail.com	224	224	2B	Pietracorbara	662	2B224	42.8446413084	9.43811996376
3	Mondoloni	Andria	15	andria.mondoloni-15@mail.com	284	284	2A	Sollacaro	353	2A284	41.7417710939	8.88010382989
4	Mattei	Julien	24	julien.mattei-7@supermail.fr	13	13	2B	Alzi	23	2B013	42.3126463350	9.31475465640
5	Marcaggi	Albert	20	albert.marcaggi-7@mymail.com	64	64	2A	Cardo-Torgia	33	2A064	41.8528771718	8.97204670982
6	Orsini	Paolo	29	paolo.orsini-8@mymail.fr	69	69	2B	Casabianca	93	2B069	42.4472062738	9.37216855737
7	Tasso	Saveriu	25	saveriu.tasso-13@supermail.fr	182	182	2B	Occhiatana	185	2B182	42.5928985700	9.00608149281
8	Cesari	Sophie	38	sophie.cesari-	300	300	2A	San-Gavino-di-	1131	2A300	41.6806017168	9.23310910423

La jointure crée en quelque sorte une **table virtuelle**, de laquelle on peut extraire les informations souhaitées.

On peut récupérer seulement le nom, le prénom et le nom de la commune de chaque utilisateur avec la requête suivante :

```
SELECT utilisateurs.nom, utilisateurs.prenom, commune.nom_commune
FROM utilisateurs
JOIN communes
  ON communes.code_commune = utilisateurs.code_commune
```

Écrire le nom de la table devant le nom de la colonne sélectionnée permet de lever toute ambiguïté qui pourrait intervenir lorsque deux colonnes ont le même nom.
On peut cependant utiliser des alias avec le mot-clé `AS` afin d'alléger un peu la requête :

```
SELECT U.nom, U.prenom, C.nom_commune
FROM utilisateurs AS U
JOIN communes AS C
  ON C.code_commune = U.code_commune
```

Comme toujours, on peut utiliser des critères (portant sur une table ou l'autre) avec `WHERE` , trier avec `ORDER BY` ...

```
-- Sélection de tous les utilisateurs habitant à Ajaccio
SELECT U.*
FROM utilisateurs AS U
JOIN communes AS C
  ON C.code_commune = U.code_commune
WHERE C.nom_commune = 'Ajaccio'

-- Sélection des utilisateurs d'au moins 30 ans
-- situés sur une commune de plus de 1000 habitants
SELECT U.*
FROM utilisateurs AS U
JOIN communes AS C
  ON C.code_commune = U.code_commune
WHERE C.population >= 1000 AND U.age >= 30
```

Question 12 Écrire une requête SQL permettant de sélectionner :

1. les utilisateurs de Corse-du-Sud
2. la moyenne d'aĝe des utilisateurs de Haute-Corse
3. le nom de la commune la moins peuplée dans laquelle se trouve au moins un utilisateur

Lorsqu'il existe plusieurs correspondances, alors la table virtuelle de jointure contient une ligne pour chaque correspondance.

```sql
-- Loisirs des utilisateurs
SELECT U.nom, U.prenom, L.intitule
FROM utilisateurs AS U
JOIN loisirs AS L
  ON L.id_utilisateur = U.id
```

nom	prenom	intitule
Orsoni	Françoise	Pêche
Orsoni	Françoise	Cuisine
Filippi	Sophie	Cinéma
Filippi	Sophie	Chant
Filippi	Sophie	Jardin
Filippi	Sophie	Musique
Filippi	Sophie	Spéléologie
Filippi	Sophie	Théâtre
Mondoloni	Andria	Animaux

```sql
-- Noms et prénoms des utilisateurs qui aiment la pêche
SELECT U.nom, U.prenom
FROM utilisateurs AS U
JOIN loisirs AS L
  ON L.id_utilisateur = U.id
WHERE L.intitule = 'Pêche'
```

Question 13 Écrire une requête SQL permettant de sélectionner :

1. l'âge moyen des utilisateurs qui aiment la chasse
2. le nombre d'utilisateurs de Corse-du-Sud qui aiment la pêche (double jointure !)

▷ ***Et si un utilisateur n'a renseigné aucun loisir ?***

Dans ce cas, il n'apparaîtra pas dans la table virtuelle, car nous faisons ici une jointure **interne**.
Pour faire apparaître les lignes ne présentant aucune correspondance, il faut faire une jointure **externe**.
On utilise alors la syntaxe `LEFT JOIN` à la place de `JOIN`, pour signifier que l'on conserve toutes les entrées de la table de gauche (celle désignée par `FROM`).

Lorsqu'il n'y a pas de correspondance, on joint à la ligne de la table « de gauche » une série de `NULL` pour chaque colonne de la table de droite.

4.4 Exercices

Exercice 01 ***Nombre d'utilisateurs par commune***

1. Quelle requête SQL permet de sélectionner la liste des noms de communes associés au nombre d'utilisateurs de cette commune ?
2. Modifier la requête précédente afin d'obtenir :
   (a) Le nombre de communes possédant plus de 5 utilisateurs
   (b) Le nom des communes ne possédant aucun utilisateur

Exercice 02 ***Distance entre deux villes***

Pour calculer la distance entre deux points A et B du globe dont on connaît les coordonnées géographiques, on dispose de la formule mathématique suivante (un peu compliquée certes) :

$$D = R_T \times \arccos(\sin\varphi_A \sin\varphi_B + \cos\varphi_A \cos\varphi_B \cos(\lambda_B - \lambda_A))$$

où φ_A et φ_B sont les latitudes des points A et B, λ_A et λ_B leurs longitudes et $R_T = 6378$km est le rayon de la Terre.
Avec cette formule et notre table *communes*, nous sommes capables de calculer la distance entre deux villes de Corse !

1. Entrer la requête SQL suivante :

```sql
SELECT C1.*, C2.*
FROM communes AS C1
JOIN communes AS C2
```

   Décrire les lignes retournées par cette requête, et préciser leur nombre.
2. Modifier la requête précédente pour ne conserver qu'une ligne, contenant les coordonnées des villes d'Ajaccio et de Bastia :

nom_commune	latitude	longitude	nom_commune	latitude	longitude
Ajaccio	41.9347926638	8.70132275974	Bastia	42.6864768806	9.42502133338

3. MySQL permet de faire des calculs, comme le montrent les requêtes suivantes (que vous testerez par vous-même !) :

```sql
-- Somme difficile !
SELECT 1 + 1 -- Affiche 2, sans surprise

-- Un peu de trigonométrie
SELECT COS(PI()) -- Affiche -1, sans surprise également (?)

-- Conversion Dégrés -> Radians
SELECT RADIANS(41.5) -- 41,5° converti en radians
```

La liste des fonctions mathématiques que l'on peut utiliser est ici : `https://sql.sh/fonctions/mathematiques`

Quelle formule SQL permet de calculer la distance entre deux villes données, en fonction des colonnes `C1.latitude`, `C1.longitude`, `C2.latitude` et `C2.longitude` ? (ne pas donner la requête entière, seulement le calcul)

4. Donner la requête SQL complète permettant de calculer la distance entre les villes d'Ajaccio et Bastia.
5. Donner la requête SQL permettant de sélectionner les deux villes de Corse les plus éloignées.

Exercice 03 ***Utilisateurs pas trop loins...***

Le but de cet exercice est de déterminer, pour un utilisateur donné, la liste des utilisateurs situés dans un périmètre donné.

1. En reprenant ce qui a été fait dans l'exercice précédent, écrire une requête SQL qui retourne la liste des communes situées à moins de 10km d'une ville donnée, par distance croissante.

   Par exemple, voici les communes à moins de 10km d'Ajaccio :

nom_commune	distance
Villanova	4.1300329440122665
Alata	4.97547103498113
Appietto	8.068953383761606
Afa	9.743294367188474

   ☞ *Il sera nécessaire de remettre le calcul de la distance dans la clause* `WHERE` *...*

2. À l'aide d'une jointure bien faite, donner la liste des communes situées à moins de 10km d'**un utilisateur donné**, par distance croissante.

   Par exemple, voici les communes à moins de 10km de l'utilisateur n°352, habitant la plus belle commune de Corse.

   ☞ *On précisera l'utilisateur par son identifiant, par exemple :* `WHERE utilisateur.id = 352`

nom_commune	distance ▲ 1
Zoza	1.4439641341185894
Mela	3.0539995904365225
Cargiaca	3.8978310904633373
Loreto-di-Tallano	3.984401429416271
Sainte-Lucie-de-Tallano	4.7411924896084345
Zérubia	4.94441565013561
Sorbollano	5.606100556722337
Olmiccia	5.686416734438421
Santa-Maria-Figaniella	5.945452428448994
Levie	7.16682330272357
Fozzano	7.514985690795058
Carbini	7.851971448188419
Serra-di-Scopamène	8.221452212284914
Arbellara	8.433292093488106
Granace	9.374235435224449

3. Donner la requête SQL permettant de donner la **liste des utilisateurs** dans un rayon de 10km d'**un utilisateur donné**.

   Voici le résultat pour notre ami 352 :

id	nom	prenom	nom_commune	distance ▲ 1
168	Poggi	Alain	Mela	3.0539995904365225
109	Rossi	Joséphine	Mela	3.0539995904365225
254	Rossi	Stella	Loreto-di-Tallano	3.984401429416271
240	Orsini	Josette	Loreto-di-Tallano	3.984401429416271
118	Appietto	Marie-Jeanne	Sainte-Lucie-de-Tallano	4.7411924896084345
326	Bartoli	Marie	Zérubia	4.94441565013561
436	Nicolai	Jean	Zérubia	4.94441565013561
266	Serreri	Joséphine	Sorbollano	5.606100556722337
405	Morazzani	Paolo	Sorbollano	5.606100556722337
79	Graziani	Livia	Sorbollano	5.606100556722337
386	Serra	Saveriu	Sorbollano	5.606100556722337
323	Colonna	Jeanne	Olmiccia	5.686416734438421
442	Castelli	Livia	Olmiccia	5.686416734438421
200	Renucci	Aurélie	Santa-Maria-Figaniella	5.945452428448994
176	Cardi	Josette	Levie	7.16682330272357
400	Filippi	Stéphane	Fozzano	7.514985690795058
280	Battesti	Élodie	Serra-di-Scopamène	8.221452212284914
463	Cardi	Julien	Arbellara	8.433292093488106

Sources

- *Données sur les communes de Corse (2014)*
  `https://www.data.gouv.fr/fr/datasets/population-de-corse-par-commune-2014/`
- *Quelques précisions sur la formule de l'exercice 1...*
  `https://geodesie.ign.fr/contenu/fichiers/Distance_longitude_latitude.pdf`