# Programmation web avancée - TD 2 - 2h

Université Lumière Lyon 2 - 2023/2024

M1 Informatique

## 1 Formulaire et BDD

Dans cet exercice, vous allez programmer en *PHP* l'affichage et le traitement d'un formulaire d'inscription suivant le modèle de la figure 1. À la validation du formulaire, votre script gérera les éventuelles erreurs suivantes : formulaire incomplet, présence de l'e-mail dans la BDD, présence de la fiche complète dans la BDD. En présence d'une erreur, un message adapté sera affiché. Sinon, la fiche sera ajoutée à la table `membre` de la BDD. Vous ferez en sorte de gérer le maintien du formulaire, c'est-à-dire qu'en cas d'erreur, les champs renseignés seront préservés lors du nouvel affichage du formulaire.

Par ailleurs, vous ferez en sorte d'afficher sous le formulaire une table HTML contenant l'ensemble des membres inscrits dans la BDD (cf figure 1).

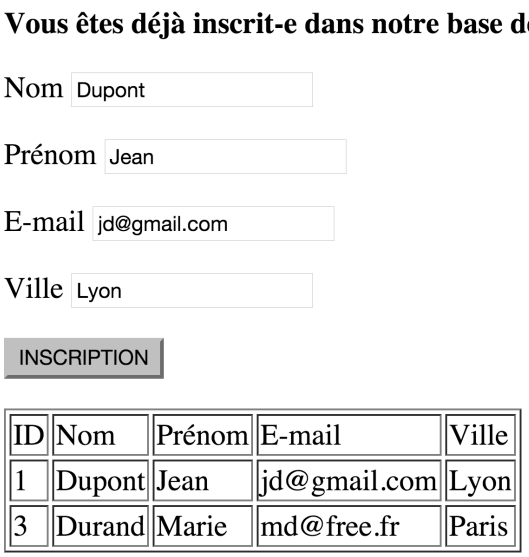


FIGURE 1 – Le formulaire d'inscription et la table des inscrits

## 2 Recherche d'anagrammes

Cet exercice va consister à programmer en *Javascript* une recherche d'anagrammes (nom féminin, contrairement à gramme ou électrocardiogramme, la vie est décidément mal faite). Étant donné un lexique des mots français (tableau indicé `aLexique` défini dans le fichier `ex2_dict.js` fourni) et un ensemble de lettres saisi par l'internaute (par exemple les lettres de son prénom et de son nom), il s'agit de trouver tous les mots du lexique formés exactement, dans un ordre quelconque, des lettres fixées par l'internaute.

Avant d'aller plus loin, commencez par prendre un moment pour réfléchir par vous-même à une méthode efficace (c-à-d avec une complexité algorithmique raisonnable) pour résoudre ce problème.

...

L'idée de base de la méthode que nous vous proposons d'implémenter est d'associer une clé à chaque mot du lexique (concrètement, une chaîne de caractères) telle que deux anagrammes (c-à-d deux mots constitués des mêmes lettres dans des ordres différents) possèdent la même clé.

Vous commencerez donc par écrire la fonction `fCalculeCle(sMot)` prenant en paramètre une chaîne de caractères, et retournant sa clé associée, à savoir la chaîne constituée de toutes les lettres de `sMot` classées par ordre alphabétique, après conversion en minuscules, suppression des éventuels caractères autres que des lettres (tirets, apostrophes, *etc.*), et remplacement des caractères accentués (*àâäçéèêëîïôöùûüÿ*) par leur version non accentuée (par exemple, les *à* deviennent des *a*, les *ç* des *c*, *etc.*). En effet, traditionnellement, la recherche d'anagrammes ignore les accents.

Écrivez maintenant la fonction `fPrepareDict()` constituant, à partir du lexique fourni, un tableau associatif (ou dictionnaire, ou encore table de hachage) dont les étiquettes sont les clés définies précédemment et les valeurs, des tableaux (indicés) contenant tous les mots du lexique associés à la clé considérée. Par exemple, la valeur associée à la clé `"aeimr"` sera le tableau `["aimer", "maire", "marie", "marié", "ramie"]` (une plante de la famille des *Urticacées*, vous aurez au moins appris ça aujourd'hui).

Vous pouvez maintenant écrire la fonction `fAnagrammesSimples(sCle, dDict)`, qui prend en paramètres une clé et le dictionnaire créé par la fonction précédente, et qui affiche les anagrammes correspondant à `sCle` (ou un message adapté s'il n'en existe pas).

Testez votre code de l'une des deux manières suivantes :

— en utilisant une page *PHP* qui affiche un formulaire pour la saisie des lettres, et réalise son traitement, consistant simplement à déclencher le script *JS* en lui *transmettant* le résultat de la saisie ;
— en utilisant la fonction *JS* `prompt()` pour la saisie.

En pratique, si le nombre de lettres saisies dépasse la dizaine, il y a peu de chances qu'une anagramme existe (bien sûr, cela dépend aussi de la fréquence dans la langue française des lettres choisies). Vous allez donc maintenant écrire un algorithme permetttant de trouver toutes les *paires* de mots du lexique dont la réunion des lettres correspond aux lettres fixées par l'internaute.

Là encore, avant de continuer la lecture, réfléchissez à la manière dont vous aborderiez ce problème.

...

Nous vous proposons d'implémenter l'algorithme suivant : étant donnée la clé `sCle` associée aux $n$ lettres fixées par l'internaute, on va parcourir le dictionnaire à la recherche des clés de longueur 1 à $n-1$ *incluses* dans `sCle`. Pour chaque clé `sC` trouvée, si le résidu (c-à-d les lettres de `sCle` qui ne sont pas dans `sC`) forme lui aussi une clé du dictionnaire, on a trouvé une solution au problème.

Écrivez la fonction `fIncluse(s1, s2)` qui prend 2 clés en paramètres et retourne un tableau (indicé) contenant un booléen dans la première case (`true` si `s1` est incluse dans `s2`, `false` sinon) et, en cas d'inclusion, le résidu (`s2` privée des lettres de `s1`) dans la deuxième case.

Il ne vous reste plus qu'à écrire la fonction `fAnagrammesDoubles(sCle, dDict)`, qui prend en paramètres une clé et le dictionnaire, et qui affiche les anagrammes doubles correspondant à `sCle` (ou un message adapté s'il n'en existe pas).

Question subsidiaire : pourquoi est-ce une bonne idée de réaliser cette recherche d'anagrammes en *JS* plutôt qu'en *PHP* ?

# 3 Annexe : Utilisation de *MySQL*

## 3.1 Connexion à la BDD

La fonction *PHP* `mysqli_connect()` permet de se connecter à la base. Elle prend en paramètres l'adresse du serveur *MySQL*, le login et le mot de passe de l'utilisateur et le nom de la base avec laquelle on souhaite travailler. Elle retourne un identifiant de connexion non nul, ou bien 0 si la connexion a échoué. On écrira donc par exemple :

```
$cnx = mysqli_connect('localhost', 'mon_login', 'mon_mdp', 'ma_base');
```

## 3.2 Envoi de requêtes à la BDD

La fonction *PHP* `mysqli_query()` permet de passer un ordre à la base. Elle prend en paramètres l'identifiant de connexion et une chaîne de caractères contenant la requête *MySQL*. Suivant la nature de la requête, elle retournera soit un code d'erreur sous forme de booléen (requêtes de création ou suppression de table, d'insertion, modification ou suppression dans une table), soit les données extraites (requêtes d'extraction). On écrira donc par exemple :

```
mysqli_query($cnx, "DROP TABLE client") or die("Echec de la suppression de la table client");
```

ou bien :

```
$data = mysqli_query($cnx, "SELECT * FROM article");
```

N.B. : le langage *MySQL* n'est pas sensible à la casse, mais l'usage veut que l'on écrive les requêtes *MySQL* en majuscules pour les distinguer plus facilement du code *PHP* (sauf pour les noms de table ou de champ, qui eux sont sensibles à la casse).

## 3.3 Syntaxe des principales requêtes MySQL

### 3.3.1 Création de table

On utilise la syntaxe `CREATE TABLE` suivie du nom de la table et d'une parenthèse dans laquelle on énumère les champs avant de spécifier la clé primaire (syntaxe `PRIMARY KEY()`). Pour chaque colonne de la table, on écrit le nom du champ suivi de son type de données. Les principaux types de données disponibles en *MySQL* sont : `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT` et `BIGINT` (entiers) ; `FLOAT`, `DOUBLE` (flottants) ; `CHAR()` et `VARCHAR()` (chaînes de taille fixe ou variable plafonnées à 255 caractères) ; `MEDIUMTEXT`, `TEXT` et `LONGTEXT` (chaînes de taille variable plus volumineuses) ; `MEDIUMBLOB`, `BLOB` et `LONGBLOB` (contenus binaires de taille variable). On pourra également utiliser les mots-clés `AUTO_INCREMENT` pour laisse *MySQL* gérer une colonne d'entiers (typiquement la clé primaire de la table), `UNSIGNED` pour restreindre un type numérique à l'intervalle positif, ou encore `IF NOT EXISTS` (avant le nom de la table) pour demander la création uniquement lorsque la table n'existe pas déjà dans la base. On écrira donc par exemple :

```
mysqli_query($cnx, "CREATE TABLE IF NOT EXISTS client (id INT UNSIGNED AUTO_INCREMENT,
                                nom VARCHAR(128), age TINYINT UNSIGNED, code_postal CHAR(5),
                                email VARCHAR(255), commentaire TEXT, PRIMARY KEY(id))");
```

### 3.3.2 Suppression de table

On utilise la syntaxe `DROP TABLE` suivie du nom de la table. Par exemple :

```
mysqli_query($cnx, "DROP TABLE client");
```

### 3.3.3 Insertion dans une table

On utilise la syntaxe `INSERT INTO` suivie du nom de la table, d'une première parenthèse contenant les noms de champs que l'on souhaite remplir, du mot-clé `VALUES`, et d'une seconde parenthèse contenant les valeurs que l'on souhaite donner à ces champs. Attention, en *MySQL*, les valeurs de type chaîne de caractères doivent **nécessairement** être encadrées d'apostrophes (et non pas de guillemets). Il est possible d'encadrer également les valeurs numériques, mais ce n'est pas indispensable. Par exemple, on écrira :

```
mysqli_query($cnx, "INSERT INTO client (nom, age, email) VALUES ('Dupont', 27, 'jd@gmail.com')");
```

N.B. : il est possible d'ajouter plusieurs enregistrements à une table en une seule requête. Pour cela, on écrira après le mot-clé `VALUE` autant de parenthèses que nécessaire, séparées par des virgules. Chaque parenthèse contiendra les valeurs de champ d'un enregistrement.

### 3.3.4 Suppression dans une table

On utilise la syntaxe `DELETE FROM` suivie du nom de la table et d'une éventuelle clause `WHERE`. Attention, en l'absence de clause `WHERE`, **toutes** les lignes de la table seront supprimées.

La clause `WHERE`, utilisée pour la suppression, mais aussi la modification et l'extraction d'enregistrements, sert à spécifier les lignes de la table impactées par la requête. Elle permet d'exprimer une condition simple ou composée (à l'aide des opérateurs `AND` et `OR`) portant sur la valeur des champs. En plus des opérateurs de comparaison classiques (`= != < >= > <=`), *MySQL* propose un opérateur `LIKE` permettant d'introduire des conditions avec le joker `_` qui remplace un caractère quelconque exactement, et le joker `%` qui remplace une séquence quelconque de caractères. Voici quelques exemples de suppression avec des clause `WHERE` différentes :

```
// Supprime l'enregistrement numéro 247 :
mysqli_query($cnx, "DELETE FROM client WHERE id = 247");
// Supprime les 10 premiers enregistrements :
mysqli_query($cnx, "DELETE FROM client WHERE id <= 10");
// Supprime les enregistrements 25 à 30 :
mysqli_query($cnx, "DELETE FROM client WHERE id >= 25 AND id <= 30");
// Supprime tous les clients lyonnais :
mysqli_query($cnx, "DELETE FROM client WHERE ville = 'Lyon');
// Supprime tous les clients dont le nom commence par "Dupon"
// suivi d'un unique caractère quelconque (par ex. "Dupont" et "Dupond") :
mysqli_query($cnx, "DELETE FROM client WHERE nom LIKE 'Dupon_');
// Supprime tous les clients dont le champ commentaire contient "machin"
mysqli_query($cnx, "DELETE FROM client WHERE commentaire LIKE '%machin%');
```

### 3.3.5 Modification dans une table

On utilise la syntaxe `UPDATE` suivie du nom de la table, du mot-clé `SET`, et d'une série de couple `champ = valeur` séparés par des virgules. Les valeurs de ces couples viendront remplacer les valeurs courantes des champs correspondants. Attention, comme pour le `DELETE`, une clause `WHERE` est généralement attendue faute de quoi les modifications seront réalisées sur **toutes** les lignes de la table! Voici deux exemples de modification :

```
 // Modifie le champ email de l'enregistrement numéro 247 :
mysqli_query($cnx, "UPDATE client SET email = 'toto@free.fr' WHERE id = 247");
// Modification à réaliser si tous les clients niçois déménageaient à Lyon :
mysqli_query($cnx, "UPDATE client SET ville = 'Lyon', code_postal = '69001' WHERE ville = 'Nice');
```

### 3.3.6 Lecture dans une table

Pour extraire des enregistrements d'une table, on utilise la syntaxe `SELECT FROM` suivie du nom de la table. Dans sa forme la plus précise, on spécifie les champs à extraire entre le `SELECT` et le `FROM`. Sinon, on

peut aussi utiliser l'étoile * pour obtenir tous les champs. Une requête SELECT peut être suivie d'une clause WHERE pour récupérer uniquement une partie des lignes de la table. Sinon, elle retournera l'intégralité des lignes. Voici quelques exemples de requêtes d'extraction :

```
// Extrait l'intégralité de la table client :
$data = mysqli_query($cnx, "SELECT * FROM client");
// Extrait les champs nom et email de toutes les lignes de la table client :
$data = mysqli_query($cnx, "SELECT nom, email FROM client");
// Extrait les champs nom et email des clients lyonnais uniquement :
$data = mysqli_query($cnx, "SELECT nom, email FROM client WHERE ville = 'Lyon'");
```

N.B. : notez que cette fois la valeur de retour de mysqli_query() est stockée dans une variable en vue de l'exploitation ultérieure des données extraites.

## 3.4 Accès aux données retournées par une requête d'extraction

En cas d'extraction (syntaxe SELECT), le résultat retourné par la fonction mysqli_query() n'est pas directement exploitable. Il s'agit d'un ensemble de lignes dont le codage interne nous est inconnu. Pour récupérer le contenu des lignes extraites, on passe donc par la fonction *PHP* dédiée mysqli_fetch_row(). Cette fonction prend en paramètre les données extraites et retourne un tableau indicé contenant les valeurs des champs de la première ligne extraite. De plus, cette fonction déplace un pointeur interne, de sorte qu'un second appel retournera les valeurs des champs de la seconde ligne extraite, et ainsi de suite. Lorsque le pointeur interne a atteint la fin des lignes extraites, tout appel supplémentaire à mysqli_fetch_row() retournera la valeur booléenne false. C'est pourquoi la récupération des lignes extraites pourra prendre la forme d'une boucle de ce type :

```
$data = mysqli_query($cnx, "SELECT nom, email FROM client");
$ligne = mysqli_fetch_row($data);         // Récupération de la première ligne extraite
while ($ligne) {                          // Tant qu'on n'a pas atteint la fin de l'extraction :
    echo "<p>$ligne[0] : $ligne[1]</p>";     // Traitement des données (ici simple affichage)
    $ligne = mysqli_fetch_row($data);         // Passage à la ligne suivante
}
```

Cette boucle est parfois condensée de la manière suivante (moins lisible mais plus compacte) :

```
$data = mysqli_query($cnx, "SELECT nom, email FROM client");
while ($ligne = mysqli_fetch_row($data)) {
    echo "<p>$ligne[0] : $ligne[1]</p>";
}
```

Notez pour finir qu'il existe une alternative à la fonction mysqli_fetch_row(). Il s'agit de la fonction mysqli_fetch_assoc(), qui fait la même chose mais retourne un tableau associatif au lieu d'un tableau indicé. Dans ce tableau, les cases ont pour étiquettes le nom des champs de la table à la place d'indices entiers. L'exemple précédent pourra donc s'écrire aussi de la manière suivante :

```
$data = mysqli_query($cnx, "SELECT nom, email FROM client");
while ($ligne = mysqli_fetch_assoc($data)) {
    echo "<p>${ligne['nom']} : ${ligne['email']}</p>";
}
```

N.B. : attention, pour accéder à une case de tableau associatif à l'intérieur d'une chaîne "...", il est indispensable de protéger le nom de variable par des accolades. (Sinon, il est bien sûr toujours possible d'utiliser une concaténation.)