\*\*\*\* Mini projet SGBD En PHP + fichiers \*\*\*\*

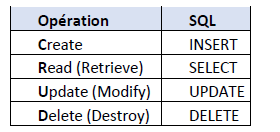
Un SGBD est un logiciel qui prend en charge la structuration, le stockage, la mise à jour et la maintenance d'une base de données. Il est l'unique interface entre les informaticiens et les données (définition des schémas, programmation des applications), ainsi qu'entre les utilisateurs et les données (consultation et mise à jour).

Exemples de SGBD :

* • Oracle est un SGBD relationnel (et relationnel-objet dans ses dernières versions) très reconnu pour les applications professionnelles.
* • PostgreSQL est un SGBD relationnel et relationnel-objet très puissant qui offre une alternative *open source* aux solutions commerciales comme Oracle ou IBM.
* • MySQL est un SGBD relationnel libre (licence GPL et commerciale), simple d'accès et très utilisé pour la réalisation de sites Web dynamiques. Depuis la version 4 MySQL implémente la plupart des fonctions attendues d'un SGBD relationnel.
* • Access est un SGBD relationnel Microsoft, qui offre une interface graphique permettant de concevoir rapidement des applications de petite envergure ou de réaliser des prototypes.

Dans ce mini projet, nous devons créer une simulation de SGBD en php qui permet de gérer les opérations de base les opérations CRUD et la création de tables.

L'acronyme CRUD se réfère à la majorité des opérations implémentées dans les bases de données relationnelles.



Nos tables seront enregistrées dans une base de données (répertoire [votre dossier git]/BDD), dans ce répertoire nous trouverons le fichier config.ini qui contient les informations de configuration de notre BDD ainsi que les comptes d’authentification.

Dans notre base de données, nous allons enregistrer des fichiers avec l’extension dwwm. Chaque fichier doit contenir un entête qui représente les champs de la table, ces derniers sont séparés par des point-virgule.

Les valeurs de chaque colonne ne peut pas dépasser 25 caractères.

Au lancement de l’application, l’utilisateur doit s’authentifier,

si le login/mot de passe sont incorrecte 🡪 Affichage d’un message d’erreur et redemander le login

Sinon, Affichage de la console d’exécution des requêtes SQL.

SQL :>create table personne(nom,prenom,telephone);

Le programme tournera en boucle tant que le user ne taperas pas « quit »

Ce système doit fournir les fonctionnalités suivantes :

**1- Création de table**

Syntaxe USER : **CREATE TABLE** nomTable(titreColonne1, titreColonne2,titreColonne3,tite) ;

Cette commande doit commencer par le mot clé create suivit du mot clé table et puis le nom de la table avec la liste des champs de cette table.

Si la syntaxe n’est pas respectée🡪 Message d’erreur

Lorsque je lance cette commande, le système doit vérifier si la table(fichier) n’existe pas , sinon , il doit afficher un message d’erreur.

**2- Insertion de données dans une table**

Syntaxe : **INSERT INTO** nomTable **VALUES**('valeur 1', 'valeur 2') ;

Cette commande permet d’insérer une nouvelle ligne dans le fichier nomTable. Le nombre de valeurs passées entre parenthèse doit correspondre au nombre de champs du fichier nomTable.

Si la syntaxe n’est pas respectée🡪Message d’erreur

Si la table n’existe pas🡪 Message d’erreur

Si le nombre de valeurs ne correspond pas au nombre de champs🡪 Message d’erreur

**3- Affichage de données**

**Syntaxe 1** : **SELECT \* FROM** nomTable;

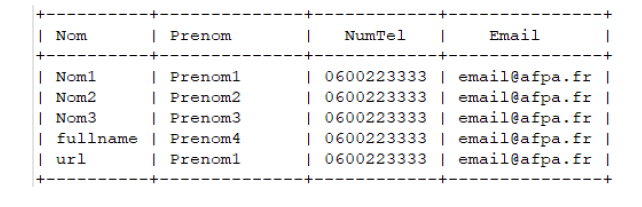
Cette requête SQL va **sélectionner** (SELECT) tous les champs **provenant** (FROM) de la table appelée “nomTable”.

Au lancement de cette commande, si la table existe le programme affichera tout le contenu de la table nomTable (cf exemple d’affichage ci-dessous).

Si la table n’existe pas, le programme affiche un message d’erreur indiquant que la table n’existe pas.

Si la syntaxe n’est pas correcte, le programme affiche un message d’erreur.

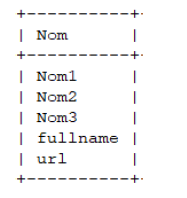
Exemple d’affichage d’une table ayant 4 colonnes :



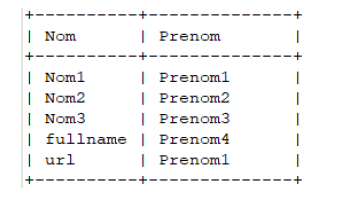
**Syntaxe 2 : SELECT** nom\_du\_champ **FROM** nomTable;

Cette requête SQL va **sélectionner** (SELECT) le champ “nom\_du\_champ” **provenant** (FROM) de la table appelé “nomTable”.

Exemple 1 : SELECT Nom from nomTable ;



Exemple 2 : SELECT Nom,Prenom from nomTable ;



BONUS

**Syntaxe 3 : SELECT \* FROM** nomTable **Order by** nom\_du\_champ **ASC;**

Cette requête SQL va **sélectionner** (SELECT) tous les champs **provenant** (FROM) de la table appelée “nomTable” et les affiche ordonnés par nom\_du\_champ.

**ASC** : Ordre croissant

**DESC** : Ordre décroissant

**4- Mise à jour de données**

La commande UPDATE permet d’effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications.

**Syntaxe 1 : UPDATE** nomTable **SET** nom\_du\_champ = 'nouvelle valeur';

Cette syntaxe permet d’attribuer une nouvelle valeur à la colonne nom\_du\_champ pour toutes les lignes de la table nomTable.

**Syntaxe 2 : UPDATE** nomTable **SET** nom\_du\_champ = 'nouvelle valeur', nom\_du\_champ1 = 'nouvelle valeur1' ;

Cette syntaxe permet d’attribuer de nouvelles valeurs aux champs nom\_du\_champ et nom\_du\_champ1 pour toutes les lignes de la table nomTable.

**Syntaxe 3 : UPDATE** nomTable **SET** nom\_du\_champ = 'nouvelle valeur' **WHERE** nom\_du\_champ = 'ancienne valeur'***;***

Cette syntaxe permet d’attribuer une nouvelle valeur à la colonne nom\_du\_champ pour les lignes qui respectent la condition stipulée avec WHERE. Il est aussi possible d’attribuer la même valeur à la colonne nom\_du\_champ pour toutes les lignes d’une table si la condition WHERE n’était pas utilisée.

CREATE TABLE personne(nom,prénom,numTel)

INSERT INTO personne VALUES(Desmoudt,Robin,0675190579)