I. Qu'est qu'une base de données ?

A. Définitions.

Une base de données est un **ensemble structuré et organisé de données** avec un **objectif commun** permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherches …).

Les bases de données sont généralement **relationnelles** car elles contiennent plusieurs tables mises en relation et qui se complètent.

B. SGBD.

La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le **S**ystème de **G**estion de **B**ase de **D**onnées (SGBD).

Un SGBD doit permettre l'ajout, la modification et la recherche de données. Un système de gestion de bases de données héberge généralement plusieurs bases de données.

Quelques caractéristiques des SGBD :

- **Accès aux données :** il se fait par l'intermédiaire d'un **L**angage de **M**anipulation de **D**onnées (LMD), ici **MySQL**. Il est crucial que ce langage permette d'obtenir des réponses aux requêtes en un temps « raisonnable ». Le LMD doit donc être optimisé, minimiser le nombre d'accès disques, et tout cela de façon totalement transparente pour l'utilisateur. On parle aussi de requêtes **SQL** (**S**tructured **Q**uery **L**angage).
- **Non redondance des données :** Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.
- **Cohérence des données :** Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression des données. Les contraintes d'intégrité sont décrites dans le Langage de Description de Données (LDD).
- **Partage des données :** Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment de manière transparente. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations, cela ne l'est plus quand il s'agit de modifications dans un contexte multi-utilisateurs car il faut : permettre à deux (ou plus) utilisateurs de modifier la même donnée « en même temps » et assurer un résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.
- **Sécurité des données :** Les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.
- **Unicité :** Une base bien conçue ne devra pas comporter plusieurs tables identiques.

C. Clés.

Afin d'identifier deux éléments d'une table, celle-ci devra disposer d'un identifiant unique pour chaque élément nommé **clé primaire** (**PRIMARY KEY**). Il s'agit le plus souvent d'un entier auto incrémenté (un nom de famille, par exemple, ne peut être une bonne clé primaire, car tôt ou tard, on aura deux enregistrements pour une même clé).

En plus de cette clé, un enregistrements peut comporter un autre identifiant qui le distingue de tous les autres ; ce sera une **clé secondaire**. Ex : numéro de série.

Lorsqu'une table fera référence à la clé primaire d'une autre table, cette clé sera nommée **clé étrangère** (**FOREIGN KEY**). Une clé étrangère, pour une table, n'est rien d'autre qu'une clé primaire dans une autre table.

D. Exemple.

Voici la table pilotes de la base de données jbt avec laquelle vous allez découvrir MySQL (capture à partir de phpMyAdmin) :

indice	nom	prenom	engin	generation
1	MANCHZECK	Jean	850 Commando	1
2	BRACAME	Edouard	CB 750	1
3	DUCABLE	Jean Raoul	750 H2	1
4	BRASLETTI	Guido	900 Desmo SS	1

Cette table contient 5 champs nommés : indice, nom, prenom, engin et generation.

Il est conseillé d'éviter les accents pour les noms des champs.

Le langage SQL fait la différence entre nom et Nom ; attention lors de l'écriture de vos requêtes.

Chaque ligne d'une table s'appelle un **enregistrement**. Lorsqu'une requête sur une base de données concernera plusieurs lignes, la réponse contiendra donc plusieurs enregistrements.

II. Utilisation d'une base de données.

Pour accéder en PHP aux données d'une base de données MySQL, les étapes sont :

1) $id = mysql_connect() ; Connexion au serveur qui héberge la base de données. L'identifiant de la connexion $id vaut 0 en cas d'échec.
2) mysql_select_db () ; Choix de la base de données parmi celles hébergées par le serveur ; cette fonction retourne 0 en cas d'échec.
3) $res = mysql_query () ; Envoi de requêtes MySQL. $res vaut 0 en cas d'échec ; En cas de succès, $res n'est pas directement affichable ! ! de plus, $res poura contenir plusieurs enregistrements (⇔ plusieurs lignes de différentes tables).

Boucle si la requête renvoie plusieurs réponses

4) mysql_fetch_row, mysql_fetch_array, mysql_fetch_assoc : Extraction d'une réponse de $res ; chaque méthode renvoie un tableau associatif différent.
5) Affichage de la réponse en HTML sur le navigateur d'où provient la requête.
6) mysql_close ($id) ; Déconnexion de la base de données.

Rem : vous disposez dans le cours de **TOUS** les prototypes des fonctions ci-dessus.

Les étapes 3, 4 et 5 sont indissociables, car le résultat de la requête est inexploitable tel quel. Il faut donc utiliser une des méthodes d'extraction de l'étape 4 qui renverra un des enregistrements de la réponse sous la forme d'un tableau associatif facile à afficher.

Il faudra donc 2 boucles imbriquées :

- Une qui extrait les enregistrements solutions un par un ;
- Une qui affiche le tableau associatif fourni pour chaque enregistrement.

III. Script PHP minimal de connexion.

A. Version perfectible.

```
if( $id = mysql_connect("localhost","toto","1234") ) {
        echo "Succès de connexion au serveur localhost. <BR />";
        if(mysql_select_db('eleves') ) {
                echo " Base de données eleves sélectionnée. <BR />";
           /* envoi de la requête et affichage des réponses ... */
        }  // fin du if(mysql_select_db('eleves') )
        else {
                die ("Echec de connexion à la base :".mysql_error() ); }
        mysql_close($id);
} // fin du if( $id = mysql_connect("localhost","toto","1234") )
else
        die ("Echec de connexion au serveur de base de données :" .mysql_error());
```

Quelle est l'adresse du serveur ? ----------

Quels sont les identifiants de connexion ? ----------

Quel est le nom de la base ? ----------

Que se passe-t-il en cas d'échec d'une primitive ? ----------

Inconvénients de cette façon de tester les erreurs ?

- ----------
- ----------

Solution ? ----------

B. Version améliorée.

```
if ( ($id = mysql_connect( "localhost","toto","1234") )_________ )
        die ("Echec de connexion au serveur de base de données : " .mysql_error());
echo "Succès de connexion au serveur de Base de Donnees. <BR />";

if ( (mysql_select_db('eleves') )_________ )
        die ("Echec de sélection de la base de données : ".mysql_error() );
echo "Succès de la sélection de la base de données. <BR />";

/* envoi de la requête et affichage des réponses ... */

mysql_close($id);
```

Améliorations :

- Le traitement des erreurs apparaît immédiatement après la requête dans le code ;
- Le corps du script, à savoir l'envoi de requêtes et l'affichage des résultats, sera écrit sans décalage.

💣 Sous EasyPHP, login = root, pas de mot de passe !

IV. Requête simplissime et extraction des réponses.

A. Interrogation d'une table de la base.

C'est la première chose à faire, une fois la connexion au serveur et la sélection de la base réussies. Pour récupérer tous les champs de la table pilotes, 2 méthodes :

1ère méthode :

```
$rq  = "SELECT * FROM pilotes" ;
$res = mysql_query ("SELECT * FROM pilotes", $id) ;
```

2nde méthode :

```
$table = "pilotes" ;
$rq = "SELECT * FROM ".$table ;
$res = mysql_query ($rq, $id) ;
```

La seconde méthode permet d'afficher la requête avec un `echo $rq ;` placé entre les 2ème et 3ème ligne ; très utile en cas d'échec de la requête.

Il est impératif de vérifier que la requête a réussi en testant $res :

```
if ( ! $res)
die ("Echec de la requete ; Erreur = " .mysql_error());
/* Exploitation du résultat de la requête MySQL  */
```

Le SGBD peut travailler à notre place ; en effet, on peut :

- limiter la taille de la réponse : `$rq = "SELECT nom, prenom FROM pilotes" ;`
- affiner la recherche : `$rq = "SELECT * FROM pilotes WHERE prenom = 'Jean' " ;`
- obtenir un résultat trié : `$rq = "SELECT * FROM pilotes ORDER BY nom " ;`

On peut combiner les trois filtres ci-dessus.

☠ **RAPPEL :** Pour l'instant, le résultat de la requête est inexploitable tel quel ! ☠

B. Exploitation du résultat d'une requête.

La réponse à la requête contient généralement plusieurs enregistrements de la table choisie qu'il faut extraire avant de les afficher. Deux familles de méthodes :

- mysql_fetch_row, mysql_fetch_array, mysql_fetch_assoc , qui renvoient pour chaque enregistrement un tableau associatif. Ce tableau sera différent pour chaque méthode.
- mysql_fetch_object, qui renvoie un pointeur vers un tableau associatif qui s'utilise comme une structure en C ⇒ il faut connaître les noms des champs et y accéder par **->**.

Dans les trois premiers cas, il faudra extraire chaque enregistrement du résultat de la requête dans une boucle while(). Exemple :

```
while ($enreg = mysql_fetch_row ($res) ) {
/* Affichage du tableau associatif correspondant à cette réponse
    par each + key et value ou par foreach (cf. TP1 sur les dates)*/
}.
```

V. Autre opérations simples sur les bases de données.

A. Connaître le nombre d'enregistrements de la réponse.

Avant de glapir que l'affichage ne se fait pas, encore faut-il être sûr que la réponse à la requête contient au moins un enregistrement ; en effet, il est possible que vous ayez mal énoncé votre condition et que la réponse soit vide. mysql_num_rows fournit cette information :

```
$res = mysql_query($rq, $id) ;
$nb_rep = mysql_ num_rows ($res) ;
echo "La reponse contient ".$nb_rep." enregistrements <BR />" ;
```

B. Insertion d'enregistrements dans une table d'une base.

```
$rq = "INSERT INTO pilotes (nom, prenom) VALUES ('SARRON','Christian'), ('ROBERTS','Kenny') " ;
```

Les champs dont le nom n'est pas spécifié dans les parenthèses après INTO pilotes seront remplis avec leur valeur par défaut (généralement 0 pour un champ numérique et une chaîne vide pour un champ texte).

Il est impératif de respecter l'ordre des champs de la base de données, et de fournir autant de valeurs que de champs mentionnés (ici 2).

Lors de la création de la base de données, on peut interdire à un champ de recevoir une valeur nulle ou vide. myqsl_query renverra donc 0 en signe d'erreur de requête => toutes les valeurs de retour des requêtes doivent être systématiquement testées.

Les données à insérer dans une table proviennent normalement d'un formulaire et sont obtenues par $_POST[ ] dans le script, ce qui provoque souvent des erreurs lors de la requête si tout est fait dans mysql_query. Il est alors impératif d'utiliser la méthode 2 de la page précédente !

C. Requête portant sur plusieurs tables d'une même base.

C'est pour cela que les bases de données sont dites **relationnelles**. Exemple de requête :

```
$rq = "SELECT champ1, champ2 FROM table1, table2" ;
```

Mais le résultat sera bizarre si les 2 tables contiennent toutes les deux le même champ (par exemple un champ nommé nom dans les deux tables). La réponse contiendra la valeur du champ nom de la table qui aura été lue en dernier ! La requête doit être :

```
$rq = "SELECT table1.champ1, table2.champ2 FROM table1, table2" ;
```

Et quand bien même les champs seraient tous différents au moment où vous avez créé la base de données, cette syntaxe présente l'avantage de préciser de quelle table vient quel champ. Elle vous met aussi à l'abri de l'ajout dans une table d'un champ portant le même nom qu'un autre champ d'un autre table.

On peut aussi (et surtout) avoir des conditions sur des éléments figurants dans différentes tables ; exemple :

```
$rq = "SELECT table1.champ1, table1.champ2 FROM table1, table2 WHERE table1.indice > 12 AND table1.nom = table2.nom" ;
```

D. Ajout d'un champ à une table.

Pour ajouter un champ nommé age de type entier à la table pilotes, la requête sera :

```
$rq = "ALTER TABLE `pilotes` ADD `age` INT ";
```

Si en plus on veut que ce champ soit forcément non nul et placé après le champ prenom :

```
$rq = "ALTER TABLE `pilotes` ADD `age` INT NOT NULL AFTER prenom" ;
```

E. Mise à jour d'un champ

Pour modifier le champ age dans la table pilotes d'Edouard BRACAME :

```
$rq = "UPDATE pilotes SET age = '42' WHERE pilotes.nom = 'BRACAME' " ;
```

F. Suppression d'un champ d'une table.

Pour supprimer le champ age dans la table pilotes :

```
$rq = " ALTER TABLE 'pilotes' DROP 'age' ";
```

G. Suppression d'un enregistrement d'une table.

Suppression d'un enregistrement complet (une ligne) dans la table pilotes :

- par indice : `$rq = " DELETE FROM 'jbt'.'pilotes' WHERE indice = 23 ";`
- par nom : `$rq = " DELETE FROM 'jbt'.'pilotes' WHERE nom = 'BRACAME' ";`

Remarques :

- écrire `indice = '23'` aurait échoué, car indice est un nombre et '23' une chaîne de caractères.
- L'indice étant une clé primaire, on est sûr de ne supprimer qu'une seule ligne.
- La suppression par nom supprimera **TOUTES** les lignes de la table qui contiennent ce nom ; peut être dangereux.

H. Autres opérations.

Il existe aussi des commandes SQL qui permettent d'ajouter ou de supprimer des tables entières, mais cela dépasse largement le cadre de cette introduction.