INF340 Systèmes d'information – Outils et méthodes informatiques pour le multimédia

Thibault Maillot

INF3 - Outils et méthodes informatiques pour le multimédia
INF340 Systèmes d'information

MAILLOT Thibault
maillot.thibault@gmail.com

2012/2013

Table des matières

Introduction

Ce cours a pour objectif de donner quelques bases sur la conception et la modélisation d'un système d'information. Il s'inclue dans la formation "Services et réseaux de communication". Celui-ci donne des méthodes de diffusion de données sur Internet ainsi qu'une description de leurs fonctionnements (transmission, réception et envoi de données).

Pour définir ce qu'est un système d'information, nous devons définir ce qu'est une donnée et une information. Une donnée est un ensemble de signes et un codage (e.g. 10/11/2006). Une information est une donnée à laquelle on associe un modèle d'interprétation, elle permet au receveur d'information de comprendre la donnée contenue (e.g. 10/11/2006 (en France) $\Longleftrightarrow$ 11/10/2006 (en Angleterre)). **Le système d'information permet de gérer des données, de les stocker et de les transmettre de manière intelligible.**

Dans la suite du cours, nous considérerons qu'un système d'information est un site Web couplé à un système de gestion de base de données. Le site nous permettra de traiter les données pour les convertir de manière intelligible par l'utilisateur.

Voici les points abordés dans ce cours :
- La modélisation des traitements.
- Les échanges de données entre client et serveur (navigateur/serveur).
- Les Formulaires de saisi, la transmission et l'enregistrement des données sur un serveur.
- Une méthode de diffusion de l'information (via création de page web dynamique).
- Des outils de développement associés aux serveurs Web : Liaison avec des systèmes des gestion de BdD (SGDB), PHP.
- La gestion des sessions et des authentifications d'utilisateurs, via cookies.

La mise en oeuvre de ces points se fera en trois parties :
- Etude de langages spécifiques du domaine (le SQL et le PHP).
- Mise en place de sites Web dynamiques.
- Etudes de cas pratiques du domaine : services sur réseaux (la sécurité et les erreurs de gestion de base de données à éviter).

Dans la suite, on suppose acquis le HTML et le CSS ainsi que les méthodes de construction d'une architecture de base de données.

1 Le fonctionnement

Avant de commencer à utiliser un site Web dynamique, il faut comprendre comment les données seront stockées et traitées par le système. En particulier, nous verrons comment se font les échanges entre le client (l'utilisateur potentiel) et le serveur (le gestionnaire du site). Par échange, nous entendons la transmission, la réception et l'envoi de données.

1.1 Architecture

Pour le développement et l'utilisation d'un site Web dynamique, avec gestion de données, nous avons besoin d'une architecture Web composée de quatre entités (c.f. Figure 1) :

- Un client Web : L'interface utilisateur.
- Un serveur HTTP (HyperText Transfer Protocol) : Le pont entre l'utilisateur et notre site.
- Un interpréteur PHP : Le compilateur PHP permettant de créer des pages dynamiques.
- Un SGBD (Système de Gestion de Base de Données) : Le gestionnaire de nos données.

Pour que notre page Web soit considérée comme **dynamique**, il faut que son contenu puisse changer en fonction des requêtes reçues. Celles-ci empruntent le chemin suivant :

1. Le client Web émet une requête vers un serveur HTTP.
2. Le serveur analyse la requête HTTP et, en fonction de ce qui est demandé, la traite. Si la requête n'a pas besoin de traitements spéciaux, elle est exécutée par le serveur qui renvoie une réponse au client (étape 4). S'il y a besoin de communiquer avec une base de données (BdD), l'interpréteur PHP fait le lien avec cette dernière.
3. Le SGBD fournit le résultat de la requête à l'interpréteur PHP.
4. L'interpréteur PHP génère la page Web, la fournit au serveur HTTP et, enfin, le serveur HTTP renvoie la réponse au client.

Dans la suite, nous utiliserons une solution tout en un, pour Windows : **EasyPHP**, disponible à l'adresse `http://www.easyphp.org/fr/`. D'autres solutions existent, comme WAMP, XAMPP, LAMP... EasyPHP offre un serveur HTTP (Apache), un serveur SQL (MySQL), un gestionnaire de base de données (MySQL) et un interpréteur PHP.

Il est toujours difficile de rendre son code indépendant de l'architecture. Il faut toujours s'intéresser à l'hébergeur et sa configuration. Généralement, chaque serveur dispose de ses fichiers de configuration, cependant ils ne sont pas toujours accessible (c'est le cas sur un hébergement mutualisé).

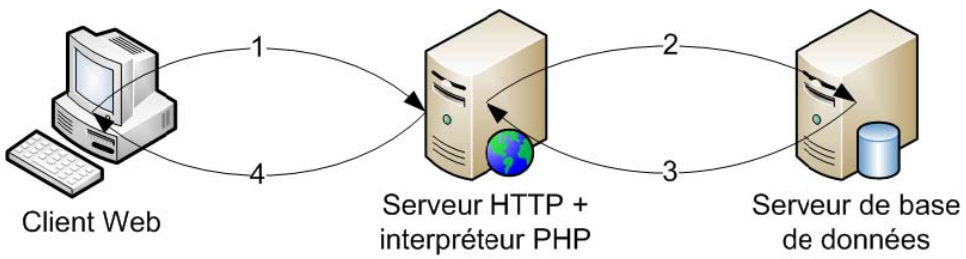

Figure 1 – Architecture du système

Lors d'un développement normal, trois phases se succèdent et trois ensembles de serveurs sont utilisés :
- **Les serveurs de développement**, dédiés au développement. L'utilisateur dispose de droits étendus.
- **Les serveurs de test** permettent d'anticiper les problèmes de déploiement, ils permettent une validation par l'équipe et par les clients.
- **Les serveurs de productions**, utilisés pour l'application finale. Aucune modification ne devrait normalement y être faite.

Par la suite, nous travaillerons sur un serveur de développement, mais il ne faut pas oublier que, pour une utilisation finale, certaines libertés ne seront pas possible.

1.2 Apache

Le serveur Web que nous utiliserons est **Apache**, c'est un serveur HTTP Open-source. Son utilisation sera transparente dans notre cas puisque nous utiliserons l'interface offerte par EasyPHP.

Le principal fichier de configuration d'Apache est `httpd.conf`. Cependant, nous ne l'utiliserons pas directement. Au besoin, nous utiliserons l'interface d'EasyPHP ainsi que les fichiers `.htaccess`. Ces fichiers sont des fichiers de configuration des serveurs Web Apache, utilisables sans droits d'administration et qui permettent de redéfinir des directives de `httpd.conf`. Les fichiers `.htaccess` sont, généralement, le seul moyen de redéfinir une configuration d'Apache offerte par votre hébergeur. On peut les utiliser pour protéger un répertoire par mot de passe, ou interdire son accès, ou encore pour changer le nom ou l'extension de la page d'index. Pour plus d'information sur les fichiers `.htaccess`, voir `http://httpd.apache.org/docs/2.3/fr/howto/htaccess.html`.

1.3 L'interpréteur PHP

Tout comme le serveur Web, l'interpréteur PHP est configurable. Le principal fichier de configuration est `php.ini`. Il permet de définir le comportement de PHP en spécifiant les extensions, les répertoires et chemins d'accès aux fichiers... Tout comme le fichier `httpd.conf`, son accès n'est pas autorisé chez un hébergeur et le seul moyen de modifier les options de PHP sont alors les fichiers `.htaccess` et `.user.ini` (voir `http://fr.php.net/manual/en/configuration.file.per-user.php`). Pour obtenir les informations sur la configuration courante de PHP, on peut utiliser la fonction `phpinfo()`.

Les informations échangées entre le serveur Web et PHP sont contenues dans des tableaux associatifs, les principaux étant `$_GET`, `$_POST` et `$_REQUEST` (par défaut, contient les données des variables `$_GET`, `$_POST` et `$_COOKIE`).

1.4 MySQL / PHPMyAdmin

Nous allons devoir rendre nos données persistantes et utiliser un SGBD. Pour cela, nous allons utiliser MySQL. Accéder à ce SGDB se fait soit par :
- Un client, comme le terminal.
- Une application Web, comme PHPMyAdmin.

Le principal fichier de configuration de MySQL est `my.ini`, mais il est plus aisé d'utiliser directement l'application Web, **PHPMyAdmin**, pour configurer le SGDB. PHPMyAdmin peut aussi être utilisé pour vérifier le résultat des scripts SQL.

2 Système de Gestion de Base de Données : utilisation du SQL

Les données à traiter doivent être normalisées. Pour cela, nous utilisons une architecture de base de données qui consiste à simplifier les informations, en mettant en avant les liaisons et les similitudes entre les données. Il faut faire attention à la redondance des informations, ainsi, pour éviter toutes incertitudes **chaque élément doit être unique** (utilisation d'un identifiant).

L'administration de la base de données sera faite via SQL. Ce langage permet, simplement, de sélectionner, créer, modifier, supprimer... des éléments. C'est ce langage qui sera présenté au cours de cette section.

Pour de plus ample informations sur la création et l'administration de base de données, voir le cours "INF 240 - Base de données" ou le web :

- `http://www.siteduzero.com/tutoriel-3-230004-les-dates-en-sql.html`
- `http://sql.1keydata.com/fr/`
- `http://sql.toutestfacile.com/`
- `http://sqlpro.developpez.com/`
- `http://sqlpro.developpez.com/cours/sqlaz/select/`
- ...

Dans cette section, nous utiliserons MySQL et PHPMyAdmin, leurs documentations sont consultables sur le net (voir `http://dev.mysql.com/doc/` ; `http://www.phpmyadmin.net/home_page/docs.php`).

2.1 Comment construire une base de données

Tout d'abord, rappelons rapidement comment construire une base de données, à partir d'un cahier des charges. Pour cela, le modèle entité-association peut être utilisé pour aboutir à une base de données correcte et bien structurée. Afin de construire un tel modèle, nous pouvons appliquer la méthode MERISE (voir `http://sqlpro.developpez.com/cours/modelisation/merise/`).

Pour appliquer cette dernière, il faut extraire les entités en jeu (i.e. les éléments majeurs), leurs attributs (i.e. les caractéristiques des entités), leurs associations (i.e. les relations entre entités) ainsi que leurs cardinalités (i.e. dénombrer le nombre d'éléments d'une entité en relations avec des éléments d'une autre entité, et inversement). Enfin, il faut veiller à créer, pour chaque entités, une clef permettant de distinguer tous les éléments de celle-ci.

Une fois ce modèle (appelé *Modèle Conceptuel des Données*) créé, il faut le rendre adaptable à l'implantation au sein d'un SGDB. En effet, ce dernier ne gère pas toutes les subtilités d'un Modèle Conceptuel des Données. Il faut donc créer un *Modèle Physique des Données*.

Voici quelques règles à respecter pour la construction de ce modèle :

1. Une table par entité.
2. Une relation père-fils (cardinalité $1 : n$ ou $0 : n$) devient une clé étrangère.
3. Une relation multiple (cardinalité $n : m$) devient une table d'association (ou de jointure).

Note :
- La cardinalité 1 : 1 est particulière. Dans ce cas, la même clef primaire peut être utilisée pour les deux tables.
- Les table d'associations peuvent avoir des attributs (par exemple la date d'une commande, le mode de paiement...).

Exemple :
Nous voulons créer une base de données permettant de gérer les classes d'IUT. Nous voudrions tout d'abord avoir accès aux effectifs des sections. Ensuite, nous aimerions avoir accès aux Nom, Genre, Date de naissance, Section, Téléphone et Moyenne de chaque étudiant. Enfin, pour pouvoir suivre l'encadrement de ces derniers, nous aimerions aussi avoir accès aux Nom, Genre, mail des enseignants de l'IUT ainsi que les Section dans lesquelles ils interviennent.

Une solutions est alors de créer quatre tables. Les trois premières nous permettent de stocker les données des sections de l'IUT, des étudiants et des enseignants. La dernière nous permet de gérer les associations existantes entre les enseignants et les sections (en effet, un enseignant peut effectuer des cours dans plusieurs sections et il y a plus d'un enseignant par section, c'est une cardinalité de type $n : m$).

Ainsi, dans la suite, nous considérerons les quatre tables 1, 2, 3, et 4.

idSection*	Section	NombreEtudiant
1	SRC	52
2	GEII	42
3	GIM	32

Table 1 – Table des sections : TABLE_SECTION

idEtudiant*	Nom	Genre	Naissance	Section	Telephone	Note
1	Nom1	M	12/12/1942	SRC	1122334400	0
2	Nom2	F	12/12/1952	SRC	2233440011	0
3	Nom3	F	12/12/1962	SRC	3344001122	0
4	Nom4	M	12/12/1972	SRC	4400112233	0
5	Nom5	M	12/12/1982	SRC	0011223344	0

Table 2 – Table des étudiant : TABLE_ETUDIANT

idEnseignant*	Nom	Genre	mail
1	Nom1	M	Nom1@univ-tln.fr
2	Nom2	F	Nom2@univ-tln.fr
3	Nom3	F	Nom31@univ-tln.fr

Table 3 – Table des enseignants : TABLE_ENSEIGNANT

idAssociation*	idEnseignant	idSection
1	1	1
2	2	1
3	2	2
4	3	1

TABLE 4 – Table des association enseignants/section : TABLE_ASSOCIATION

2.2 La définition des éléments : CREATE, ALTER, DROP

La définition des éléments permet de créer, modifier ou supprimer une base de données, une table... Il existe trois commandes principales : **CREATE**, **ALTER** et **DROP**.

La commande CREATE

Elle permet de créer une table en ajoutant, ou non, des contraintes de colonnes comme :
- **NULL / NOT NULL** : Obligation de saisi d'une valeur, ou pas.
- **DEFAULT** : Valeur par défaut.
- **COLLATE** : L'ordre des caractères à prendre en compte pour le tri.
- **PRIMARY KEY** : Défini la clef primaire (la colonne doit être NOT NULL).
- **UNIQUE** : Les valeurs sont soit unique, soit NULL.
- **CHECK** : Condition pour accepter la valeur.
- **FOREIGN KEY** : Clef secondaire - toute tentative de créer un élément qui a une valeur de clef secondaire n'existant pas dans la table de référence est impossible, tout comme supprimer un élément de la table de référence si une de ces valeurs est utilisées comme clef secondaire.

Il est aussi possible d'ajouter des contraintes sur la table elle-même, si l'on souhaite avoir une clef multi-colonne, par exemple. Les contraintes possibles sont :
- **PRIMARY KEY**
- **UNIQUE**
- **CHECK**
- **FOREIGN KEY**
- **INDEX** : Ajout d'une colonne dans l'index de table

ATTENTION :
- Pour la clause **FOREIGN KEY**, la colonne spécifiée comme référence doit être une colonne indexée (une clef ou un index) grâce à la méthode **INDEX**.
- Il se peut que les apostrophes simples de ce document ne soient pas reconnues par le gestionnaire de base de données. Dans les exemples, il sera préférable de les réécrire.

Voici un exemple de création de tables, avec une clef secondaire :

```
-- Création de la table TABLE_REFERENCE
CREATE TABLE TABLE_REFERENCE
(
        idRef     INTEGER  NOT NULL PRIMARY KEY,
        NomRef    CHAR(10),
        INDEX(NomRef) -- On l'ajoute à l'index car ce n'est pas une clef
);

-- Création de la table TABLE_SECONDAIRE
-- Cette table contient une clef secondaire
CREATE TABLE TABLE_SECONDAIRE
```

```
(
        idSec     INTEGER  NOT NULL PRIMARY KEY,
        NomRef    CHAR(10),
        Note      CHAR(50),
        FOREIGN KEY (NomRef) REFERENCES TABLE_REFERENCE(NomRef)
)

-- Insertion de données
INSERT INTO TABLE_REFERENCE  VALUES (1, "Dupont");
INSERT INTO TABLE_REFERENCE  VALUES (2, "Duval");
INSERT INTO TABLE_SECONDAIRE VALUES (1, "Dupont", "entree1");
INSERT INTO TABLE_SECONDAIRE VALUES (2, "Dupont", "entree2");

-- Test de la clef secondaire 1 : Problème car la clef secondaire n'existe pas
INSERT INTO TABLE_SECONDAIRE VALUES (3, "Henri", "essai_ajout");
-- Test de la clef secondaire 2 : Problème car la clef est encore utilisée
DELETE FROM TABLE_REFERENCE WHERE NomRef = "Dupont";
-- Test de la clef secondaire 3 : Aucun problème
DELETE FROM TABLE_REFERENCE WHERE NomRef = "Duval";
```

Rappel : Pour spécifier une valeur littérale, il faut l'entourer de guillemets simples ou doubles.

Exemple de clef multi-colonne :

```
CREATE TABLE TABLE_ETUDIANT
(
        Nom        CHAR(50) NOT NULL,
        Genre      CHAR(1) DEFAULT "X",
        Naissance  DATE,
        Section    CHAR(10),
        Telephone  CHAR(14) NOT NULL UNIQUE,
        Note       DOUBLE  CHECK(Note >= 0),
        FOREIGN KEY (Section) REFERENCES TABLE_SECTION (Section),
        CONSTRAINT clef_primaire PRIMARY KEY (Nom, Telephone)
) ENGINE = InnoDB ;
```

Nous pouvons maintenant créer les Tables 1, 2, 3 et 4 :

```
CREATE TABLE TABLE_SECTION
(
        idSection         INTEGER NOT NULL PRIMARY KEY auto_increment,
        Section           CHAR(10),
        NombreEtudiant    INTEGER,
        INDEX(Section)
);

CREATE TABLE TABLE_ETUDIANT
(
        idEtudiant INTEGER PRIMARY KEY auto_increment,
        Nom        CHAR(50) NOT NULL,
        Genre      CHAR(1) DEFAULT "X",
        Naissance  DATE,
        Section    CHAR(10),
        Telephone  CHAR(14) NOT NULL UNIQUE,
        Note       DOUBLE  CHECK(Note >= 0),
        FOREIGN KEY (Section) REFERENCES TABLE_SECTION (Section) ON DELETE CASCADE
);

CREATE TABLE TABLE_ENSEIGNANT
(
        idEnseignant     INTEGER NOT NULL PRIMARY KEY auto_increment,
        Nom               CHAR(50),
        Genre             CHAR(1) DEFAULT "X",
        Mail              CHAR(50)
);
```

```
CREATE TABLE TABLE_ASSOCIATION
(
        idAssociation     INTEGER NOT NULL PRIMARY KEY auto_increment,
        idEnseignant      INTEGER NOT NULL,
        idSection         INTEGER NOT NULL,
        FOREIGN KEY (idEnseignant) REFERENCES TABLE_ENSEIGNANT (idEnseignant) ON DELETE CASCADE,
        FOREIGN KEY (idSection) REFERENCES TABLE_SECTION (idSection) ON DELETE CASCADE
);
```

Note :
- L'utilisation d'une vérification de type **CHECK** est admise par MySQL mais n'est pas prise en compte...
- Le nom donné aux contraintes (ici les clefs étrangères) servent à les reconnaître plus facilement lors d'une erreur.
- L'utilisation de la close **ENGINE** permet de définir la méthode de gestion et d'enregistrement de la table.

Nous l'avons vu, l'utilisation de **CHECK** peut poser problème. Une solution existe et consiste à utiliser un **TRIGGER**, qui est un morceau de code se déclenchant sous certaines conditions, dont la syntaxe est :

```
CREATE TRIGGER nom_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nom_table
[FOR each row]
code
```

A noter que dans le code du **TRIGGER**, nous pouvons utiliser deux pseudos tables **OLD** et **NEW**, qui contiennent les données avant et en cours de modification.

Voici une application de **TRIGGER** :

```
-- Création du TRIGGER
-- Attention : il faut définir le délimiteur comme étant &&
CREATE TRIGGER MonTrigger BEFORE INSERT ON TABLE_ETUDIANT
FOR EACH ROW
BEGIN
        IF(New.Note<0)
                THEN SET New.Note=NULL;
        END IF;
END &&

-- Test
-- ATTENTION : il faut définir le délimiteur comme étant ;
INSERT INTO TABLE_SECTION (idSection, Section, NombreEtudiant) VALUES (NULL, "SRC", NULL);
INSERT INTO TABLE_ETUDIANT (idEtudiant, Nom, Genre, Naissance, Section, Telephone, Note)
        VALUES (100, "Nom1", "M", "1942-12-12", "SRC", 123456, -10);

-- Suppression du Trigger
DROP TRIGGER MonTrigger
```

La commande ALTER

Elle permet de modifier un élément de la base. Cette commande nous permet de :
- Supprimer une colonne.
- Supprimer une contrainte.
- Ajouter une colonne.
- Ajouter une contrainte.

– Changer le nom d'une colonne.

Par exemple :

```
-- Ajouter une clef primaire
ALTER TABLE TABLE_ETUDIANT ADD PRIMARY KEY (idEtudiant);
-- Ajouter une colonne
ALTER TABLE TABLE_ETUDIANT ADD Gender char(10)
-- Ajout d'une valeur par défaut
ALTER TABLE TABLE_ETUDIANT ALTER Gender SET DEFAULT "truc"
-- Suppression d'une valeur par défaut
ALTER TABLE TABLE_ETUDIANT ALTER Gender DROP DEFAULT
-- Supprime Gender
ALTER TABLE TABLE_ETUDIANT DROP Gender
-- Change le nom d'une colonne ainsi que son type
ALTER TABLE TABLE_ETUDIANT CHANGE Nom Nom_Prenom VARCHAR(30)
-- Ajout d'une contrainte
ALTER TABLE TABLE_ETUDIANT
        ADD CONSTRAINT CHK_DATE_NAISSANCE
        CHECK (Naissance BETWEEN "1880-01-01" AND "2020-01-01")
```

La commande DROP

Elle permet d'effectuer des suppressions de tables et est assez simple à utiliser. Son utilisation est donc à effectuer avec précaution. Voici un exemple d'application :

```
-- Supprime la table des étudiants
DROP TABLE TABLE_ETUDIANT
```

2.3 La manipulation des données : SELECT, INSERT, UPDATE, DELETE

La manipulation des données est la partie la plus importante d'un gestionnaire de base de données puisqu'elle traite les éléments sur lesquels nous souhaitons travailler. Les commandes principales sont :

– **INSERT** : Insérer des données.
– **UPDATE** : Modifier des données.
– **DELETE** : Supprimer des données.
– **SELECT** : Extraire des données.

Voici un exemple d'application des **trois premières commandes**, la dernières étant plus importante, son explication est effectuée par la suite. Le script qui suit permet la construction des données des Tables 1, 2, 3 et 4.

```
-- Ajout des lignes aux tables :
INSERT INTO TABLE_SECTION (idSection, Section, NombreEtudiant)
        VALUES (1,"SRC",52);
INSERT INTO TABLE_SECTION (idSection, Section, NombreEtudiant)
        VALUES (2,"GEII",42);
INSERT INTO TABLE_SECTION (idSection, Section, NombreEtudiant)
        VALUES (3,"GIM",32);
INSERT INTO TABLE_ETUDIANT (idEtudiant, Nom, Genre, Naissance, Section, Telephone, Note)
        VALUES (1, "Nom1", "M", "1942-12-12", "SRC", 1122334400, 0);
INSERT INTO TABLE_ETUDIANT (idEtudiant, Nom, Genre, Naissance, Section, Telephone, Note)
        VALUES (2, "Nom2", "M", "1952-12-12", "GEII", 2233440011, 0);
INSERT INTO TABLE_ETUDIANT (idEtudiant, Nom, Genre, Naissance, Section, Telephone, Note)
        VALUES (3, "Nom3", "M", "1962-12-12", "SRC", 3344001122, 0);
INSERT INTO TABLE_ETUDIANT (idEtudiant, Nom, Genre, Naissance, Section, Telephone, Note)
        VALUES (4, "Nom4", "M", "1972-12-12", "SRC", 4400112233, 0);
INSERT INTO TABLE_ETUDIANT (idEtudiant, Nom, Genre, Naissance, Section, Telephone, Note)
        VALUES (5, "Nom5", "M", "1982-12-12", "SRC", 0011223344, 0);
INSERT INTO TABLE_ENSEIGNANT (idEnseignant, Nom, Genre, Mail)
        VALUES (1, "Nom6", "M", "Nom1@univ-tln.fr");
```

```
INSERT INTO TABLE_ENSEIGNANT (idEnseignant, Nom, Genre, Mail)
        VALUES (2, "Nom7", "M", "Nom2@univ-tln.fr" );
INSERT INTO TABLE_ENSEIGNANT (idEnseignant, Nom, Genre, Mail)
        VALUES (3, "Nom8", "M", "Nom3@univ-tln.fr");
INSERT INTO TABLE_ASSOCIATION (idAssociation, idEnseignant, idSection)
        VALUES (1, 1, 1);
INSERT INTO TABLE_ASSOCIATION (idAssociation, idEnseignant, idSection)
        VALUES (2, 2, 1);
INSERT INTO TABLE_ASSOCIATION (idAssociation, idEnseignant, idSection)
        VALUES (3, 2, 2);
INSERT INTO TABLE_ASSOCIATION (idAssociation, idEnseignant, idSection)
        VALUES (4, 3, 1);

-- Pour modifier une ligne de données :
UPDATE TABLE_ETUDIANT
SET Note = 20
WHERE Section = "SRC"
AND DATEDIFF( now( ) , Naissance ) >10 ;

-- Supprimer une donnée :
DELETE FROM TABLE_ETUDIANT WHERE idEtudiant=1 ;
```

La commande SELECT

Cette commande a beaucoup d'utilisations différentes et permet l'extraction, ou la création d'informations, à partir de données existantes. Son utilisation se fait selon la syntaxe suivante :

```
SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM nom de table ou de vue (données virtuelles)
[WHERE prédicats]
[GROUP BY ordre des groupes]
[HAVING condition]
[ORDER BY liste de colonnes]
```

Dans cette syntaxe, il y a deux clauses obligatoires et quatre optionnels (entre crochets). Voici leurs significations :

- **SELECT** : Choix des colonnes à extraire.
- **FROM** : Spécifications des tables à traiter.
- **WHERE** : Filtrage des données (conditions pour apparaître dans le résultat).
- **GROUP BY** : Définition d'un sous-ensemble, généralement utilisée lorsqu'une fonction de regroupement est utilisée (e.g. **COUNT()**, **SUM()**, **GROUP_CONCAT()**).
- **HAVING** : Filtrage des résultats (e.g. après l'utilisation d'une somme, supprimer les lignes ayant un résultat inférieur à une valeur).
- **ORDER BY** : Triage des données du résultat.

Voici un premier exemple nous permettant de récupérer **toutes les informations** des étudiants de SRC **et de les trier** par ordre alphabétique.

```
SELECT   *
FROM     TABLE_ETUDIANT
WHERE    Section="SRC"
ORDER BY Nom
```

Pour obtenir toutes **les sections distinctes** des étudiants, une solution serait d'exécuter la commande :

```
SELECT distinct   Section
FROM              TABLE_ETUDIANT
ORDER BY          Section
```

Pour un résultat plus "jolie", nous pouvons **modifier l'en-tête** d'une colonne du résultat grâce à **AS** et effectuer une concaténation de chaînes de caractères grâce à la fonction **CONCAT** :

```
SELECT    CONCAT(Nom , " est en section " , Section) AS "Les étudiants"
FROM      TABLE_ETUDIANT
```

Bien entendu, les opérateurs mathématiques basiques sont aussi implémentés. Pour **calculer la moyenne** de classe, nous pouvons exécuter :

```
SELECT SUM( Note ) / COUNT( Note ) AS "Moyenne de SRC"
FROM TABLE_ETUDIANT
WHERE Section = "SRC"
```

Enfin, il est aussi possible d'**inclure de la logique** dans les requêtes SQL.

```
SELECT Nom,
        CASE Section
        WHEN "SRC" THEN Note + 10
        ELSE 0
        END AS Note_modifiées
FROM TABLE_ETUDIANT
```

A noter : Il existe des opérateurs utiles comme :
- **IN** : Permet de tester si une valeur est dans un groupe donnée.
- **BETWEEN** : Permet de tester si une valeur est comprise entre deux autres.
- **LIKE** : Permet d'effectuer des tests sur les valeurs avec plus de souplesse que **BETWEEN** ou **IN**.
- **AS** : Permet de renommer une colonne de données.
- **CONCAT** : Permet de concaténer des chaînes de caractères.
- **EXIST** : Permet de savoir si la requête retourne une ligne.
- **SUM** : Permet de sommer les données d'une colonne de résultats.
- **COUNT** : Permet d'obtenir le nombre de données d'une colonne de résultats.
- ...

Les jointures

Bien que le **SELECT** soit un outil puissant, il a besoin d'informations pour travailler. Ces données lui sont apportées par la clause **FROM**. Dans la majeure partie des extractions, les données de la base doivent être recoupées. L'utilisation de jointures, d'unions et d'intersections est indispensable dans certains cas. Pour cela plusieurs types de méthodes existent :
- La jointure interne.
- La jointure externe.
- La jointure naturelle.
- La jointure croisée.
- L'union.

Il existe une jointure par défaut, pour laquelle il n'y a pas besoin d'ajouter de mots clef supplémentaire : **la jointure croisée**. En effet, les deux commandes suivantes renvoient le même résultat :

```
-- Commande 1 : utilisation du mot clef CROSS JOIN
SELECT *
FROM TABLE_ENSEIGNANT CROSS JOIN TABLE_ETUDIANT

-- Commande 2 : sans utilisation de mot clef
SELECT *
FROM TABLE_ENSEIGNANT , TABLE_ETUDIANT
```

Le majeur problème de **la jointure croisée** est qu'elle effectue un produit cartésien des données des tables, i.e. à chaque ligne d'une table est associée toutes les lignes de la seconde. Pour avoir des résultats plus intéressant, les clauses **WHERE** ou **INNER JOIN** peuvent être utilisées, dans ce cas, une **jointure interne** sera effectuée. L'exemple suivant extrait les professeurs de chaque étudiant.

```
-- Commande 1 : utilisation du mot clef WHERE
SELECT Profs.Nom As "Enseignant", TABLE_ETUDIANT.Nom As "Etudiant"
FROM
           (
                   SELECT Nom, Section
                   FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
                   WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION .idEnseignant
                           AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
           ) As Profs ,
           TABLE_ETUDIANT
WHERE Profs.Section=TABLE_ETUDIANT.Section
ORDER BY TABLE_ETUDIANT.Nom, Profs.Nom

-- Commande 2 : utilisation du mot clef INNER JOIN
SELECT Profs.Nom As "Enseignant", TABLE_ETUDIANT.Nom As "Etudiant"
FROM
           (
                   SELECT Nom, Section
                   FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
                   WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION .idEnseignant
                           AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
           ) As Profs
       INNER JOIN TABLE_ETUDIANT
ON Profs.Section=TABLE_ETUDIANT.Section
ORDER BY TABLE_ETUDIANT.Nom, Profs.Nom
```

Note : **La jointure interne** étant la plus courante, le mot clef **INNER** peut être omis, i.e. nous pouvons utiliser la clause **JOIN** seule.

La **jointure naturelle** permet de ne pas spécifier la colonne de recoupement (c'est le gestionnaire qui recherche les colonnes identiques). Mais certaines surprises peuvent apparaître. Par exemple si nous oublions que les colonnes du nom des tables enseignants et étudiants portent le même labelle "Nom", nous pourrions être étonné du résultat de l'exemple suivant si nous voulions afficher la liste des enseignants de chaque étudiants :

```
SELECT Profs.Nom As "Enseignant", TABLE_ETUDIANT.Nom As "Etudiant"
FROM
           (
                   SELECT Nom, Section
                   FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
                   WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION .idEnseignant
                           AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
           ) As Profs
       NATURAL JOIN TABLE_ETUDIANT
ORDER BY Etudiant
```

Note : Lors des deux derniers exemples, nous avons utilisé une table intermédiaire (nommée *Profs*) pour extraire les sections dans lesquelles chaque enseignant intervient (utilisation de la table d'association).

La jointure interne a aussi ces limites. En effet, si un utilisateur souhaite obtenir la liste de **toutes** les sections avec leurs enseignants associés, la jointure interne ne fonctionnera pas toujours (c'est le cas dans notre d'exemple puisqu'il existe des sections sans enseignants) :

```
SELECT TABLE_SECTION.Section As "Section", Profs.Nom As "Enseignant"
FROM
```

```
        (
           SELECT Nom, Section
           FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
           WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION .idEnseignant
              AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
        ) As Profs
        INNER JOIN TABLE_SECTION
ON Profs.Section=TABLE_SECTION.Section
ORDER BY Section, Enseignant
```

En exécutant cette instruction, nous observons que la section "GIM" a disparu... Pas très utile si le but était de voir qu'elles étaient les sections ayant besoin d'enseignants. Pour palier à ce problème, il existe la **jointure externe** dont la syntaxe est :

```
SELECT ...
FROM   <table gauche>
        LEFT | RIGHT | FULL OUTER JOIN <table droite 1>
        ON <condition de jointure>
        [LEFT | RIGHT | FULL OUTER JOIN <table droite 2>
        ON <condition de jointure 2>]
...
```

Pour effectuer une **jointure externe**, le gestionnaire à besoin de savoir qu'elle est la méthode à utiliser, d'où l'existence du choix de mot clefs avant **OUTER JOIN**. Il existe trois méthodes différentes :

- **LEFT** : Recherche toutes les valeurs satisfaisant la condition de jointure, puis on rajoute toutes les lignes de la table se situant à gauche du mot clef **JOIN** qui n'ont pas été prises en compte au titre de la satisfaction du critère.
- **RIGHT** : Recherche toutes les valeurs satisfaisant la condition de jointure, puis on rajoute toutes les lignes de la table se situant à droite du mot clef **JOIN** qui n'ont pas été prises en compte au titre de la satisfaction du critère.
- **FULL** : Recherche toutes les valeurs satisfaisant la condition de jointure, puis on rajoute toutes les lignes de la table se situant à gauche et à droite du mot clef **JOIN** qui n'ont pas été prises en compte au titre de la satisfaction du critère, c'est une **jointure bilatérale**.

L'application de la jointure externe, à notre recherche d'enseignant manquant, peut être faite de plusieurs manières dont celle-ci :

```
SELECT TABLE_SECTION.Section As "Section", Profs.Nom As "Enseignant"
FROM
        (
           SELECT Nom, Section
           FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
           WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION .idEnseignant
              AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
        ) As Profs
        RIGHT OUTER JOIN TABLE_SECTION
ON Profs.Section=TABLE_SECTION.Section
ORDER BY Section, Enseignant
```

En plus des jointures, d'autres méthodes existent pour traiter des données et les regrouper en ensembles. La méthode **GROUP BY** en est un exemple (e.g. pour faire un comptage des étudiants par section).

Les commandes suivantes permettent de construire la table d'effectif des sections et la table du nombre d'étudiants, par section, à avoir la moyenne (en utilisant la clause **HAVING**).

```
-- Commande 1
SELECT Section, COUNT(*) AS Effectif
```

```
FROM   TABLE_ETUDIANT
GROUP  BY Section

-- Commande 2
SELECT Section, COUNT(*) AS Effectif
FROM   TABLE_ETUDIANT WHERE Note>10
GROUP  BY Section
HAVING COUNT(*) >= 1
```

Enfin, des ensembles peuvent aussi être créés par des unions, comme montré dans l'exemple suivant. Dans ce cas, les colonnes des tables à unir doivent porter les mêmes noms.

```
-- L'union
SELECT Nom FROM TABLE_ETUDIANT
UNION
SELECT Nom FROM TABLE_ENSEIGNANT
```

2.4 La gestion des droits d'accès : GRANT, REVOKE

La gestion des droits d'accès est la partie du SQL qui s'occupe de gérer les autorisations d'accès aux tables. Deux commandes principales existent :
- **GRANT** : Attribuer des droits à un utilisateur.
- **REVOKE** : Révoquer des droits.

Une utilisation simpliste de ces deux modules est présentée ici, pour plus d'informations voir le web [1]. Une utilisation plus avancée de ces options permet de gérer le nombre de requêtes et de modifications accordés à un utilisateur.

L'utilisation de la base de données est effectuée, dans la plupart des cas, par l'administrateur (root) qui a tout les droits (et généralement sans mot de passe). Les dangers d'une telle utilisation sont assez clairs et le recours à d'autres profils est recommandée, d'autant plus que la plupart des personnes utilisatrices de la base de données auront juste besoin d'avoir un accès en lecture et écriture sur certaines tables.

Pour créer un utilisateur, que l'on nommera ROBERT, avec les droits d'accès en extraction de données uniquement sur les élèves, nous pouvons exécuter :

```
GRANT SELECT
        ON TABLE_ETUDIANT
        TO ROBERT IDENTIFIED BY "lemotdepassederobert"
```

Si nous aurions voulu aussi lui donner l'accès en écriture et ceux, sur toutes les tables, nous aurions pu exécuter :

```
GRANT SELECT,INSERT
        ON *
        TO ROBERT IDENTIFIED BY "lemotdepassederobert"
```

Par contre, si nous souhaitons supprimer l'accès de la base de données de ROBERT, nous utiliserons **REVOKE** :

```
REVOKE ALL
        ON *
        FROM ROBERT
```

1. http://dev.mysql.com/doc/refman/5.0/fr/grant.html ; http://sqlpro.developpez.com/cours/sqlaz/dcl/

Afin de mieux comprendre les exemples précédent, voici les syntaxes de **GRANT** et **REVOKE** :

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
        ON {tbl_name | * | *.* | db_name.*}
        TO user [IDENTIFIED BY [PASSWORD] 'password']
                [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
                [REQUIRE
                        NONE |
                        [{SSL| X509}]
                        [CIPHER cipher [AND]]
                        [ISSUER issuer [AND]]
                        [SUBJECT subject]]
        [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR count |
                              MAX_UPDATES_PER_HOUR count |
                              MAX_CONNECTIONS_PER_HOUR count]]

REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
        ON {tbl_name | * | *.* | db_name.*}
        FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Voici une liste d'actions autorisables (ou non) avec **GRANT** et **REVOKE**, en remplaçant le terme `priv_type` de la syntaxe :

- **ALL** [PRIVILEGES] : Donne accès à tous les droits, sauf WITH GRANT OPTION.
- **ALTER** : Autorise l'utilisation de ALTER TABLE.
- **CREATE** : Autorise l'utilisation de CREATE TABLE.
- **CREATE TEMPORARY TABLES** : Autorise l'utilisation de CREATE TEMPORARY TABLE.
- **DELETE** : Autorise l'utilisation de DELETE.
- **DROP** : Autorise l'utilisation de DROP TABLE.
- **EXECUTE** : Autorise l'utilisateur à exécuter des procédures stockées (pour MySQL 5.0).
- **FILE** : Autorise l'utilisation de SELECT ... INTO OUTFILE et LOAD DATA INFILE.
- **INDEX** : Autorise l'utilisation de CREATE INDEX et DROP INDEX.
- **INSERT** : Autorise l'utilisation de INSERT.
- **LOCK TABLES** : Autorise l'utilisation de LOCK TABLES sur les tables pour lesquelles l'utilisateur a les droits de SELECT.
- **PROCESS** : Autorise l'utilisation de SHOW FULL PROCESSLIST.
- **REFERENCES** : Réservé pour le futur.
- **RELOAD** : Autorise l'utilisation de FLUSH.
- **REPLICATION CLIENT** : Donne le droit à l'utilisateur de savoir où sont les maîtres et esclaves.
- **REPLICATION SLAVE** : Nécessaire pour les esclaves de réplication (pour lire les historiques binaires du maître).
- **SELECT** : Autorise l'utilisation de SELECT.
- **SHOW DATABASES** : SHOW DATABASES affiche toutes les bases de données.
- **SHUTDOWN** : Autorise l'utilisation de `mysqladmin shutdown`.
- **SUPER** : Autorise une connexion unique même si `max_connections` est atteint, et l'exécution des commandes CHANGE MASTER, KILL `thread`, `mysqladmin debug`, PURGE MASTER LOGS et SET GLOBAL.
- **UPDATE** : Autorise l'utilisation de UPDATE.
- **USAGE** : L'utilisateur n'a pas de droits.
- **GRANT OPTION** : Autorise l'utilisateur à donner des droits à d'autres utilisateur (dangereux).

Note : Il est recommandé de ne pas donner le droit **ALTER** à un utilisateur normal. En effet, avec celui-ci, l'utilisateur pourra essayer de contourner le système de droits en renommant certaines tables.

2.5 La gestion des transactions

Le module de gestion des transaction permet, comme son nom l'indique, de contrôler la bonne exécution des **transactions** qui sont des demandes de manipulations de la base de données. L'importance de ce module vient principalement du fait que la base peut être utilisée par plusieurs personnes à la fois. Cette utilisation en parallèle peut poser de gros problèmes, par exemple le double achat d'un unique objet parce que deux utilisateurs en ont fait la demande dans un laps de temps assez proche.

Généralement, les transactions doivent vérifier les propriétés suivantes :
- **Atomicité** : Une transaction s'effectue ou non (tout ou rien), même si le système connaît une panne en cours d'exécution de la requête. Un code atomique est un programme qui s'exécute sans jamais être interrompu par un processus concourant. Il a donc l'exclusivité des ressources pendant tout le temps de son exécution.
- **Isolation** : Les transactions sont isolées les unes des autres. Il n'y a donc pas de vues partielles des données pendant toute la durée de la transaction de mise à jour.
- **Cohérence** : Le résultat ou les changements induits par une transaction doivent impérativement préserver la cohérence de la base de données. Par exemple, la concaténation de tables ne peut entraîner la présence du même identifiant : il faudra résoudre les conflits d'identifiants avant d'opérer l'union des tables.
- **Durabilité** : Une fois validée, une transaction doit perdurer, c'est à dire que les données sont persistantes même s'il s'ensuit une défaillance dans le système.

Pour répondre à ce cahier des charges, plusieurs commandes sont disponible :
- **SET TRANSACTION** : Changer les options des transactions.
- **COMMIT** : Valider l'exécution.
- **ROLLBACK** : Annuler l'exécution.
- **TRIGGER** : Créer des tâches automatisées.

L'atomicité

L'atomicité peut être faites en utilisant la déclaration d'une transaction. Il existe deux cas possibles, soit le serveur est configuré pour travailler en **AUTO COMMIT** (tous les ordres sont des transactions et toute modification est validée et enregistrée), soit le serveur ne travaille pas en **AUTO COMMIT** (c'est à l'utilisateur de valider la transaction). Pour vérifier, sous MySQL, si le serveur est en **AUTO COMMIT**, nous pouvons exécuter `SELECT @@autocommit`. Nous pouvons aussi désactiver l'**AUTO COMMIT** grâce à `SET AUTOCOMMIT=0`.

Si le serveur ne travaille pas en **AUTO COMMIT**, à chaque fin de transaction, il faut soit la valider avec **COMMIT**, soit l'annuler avec **ROLLBACK**. Voici la syntaxe à utiliser :

```
-- Le serveur fait de l'AUTO COMMIT
BEGIN TRANSACTION [nom_transaction]
...
ROLLBACK | COMMIT TRANSACTION [nom_transaction]
...
COMMIT | ROLLBACK TRANSACTION [nom_transaction]

-- Le serveur ne fait pas d'AUTO COMMIT
...
ROLLBACK | COMMIT TRANSACTION
```

```
...
COMMIT | ROLLBACK TRANSACTION

-- Exemple pour le serveur en non AUTO COMMIT sous MySQL
SET AUTOCOMMIT=0 ;

START TRANSACTION; -- Pas utile mais pratique si on oublie l'AUTOCOMMIT
SELECT @A:=SUM(Note) FROM TABLE_ETUDIANT WHERE Section="SRC";
UPDATE TABLE_ETUDIANT SET Note=@A WHERE idEtudiant=1;
COMMIT;

START TRANSACTION; -- Pas utile mais pratique si on oublie l'AUTOCOMMIT
UPDATE TABLE_ETUDIANT SET Note=30 WHERE idEtudiant=1;
SELECT @A:=Note FROM TABLE_ETUDIANT WHERE idEtudiant=1;
ROLLBACK;
```

L'isolation

L'isolation, quand à elle, est configurée grâce à la commande **SET TRANSACTION** :

```
SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITED | READ COMMITED | REPEATABLE READ | SERIALIZABLE}
```

Comme nous pouvons le voir dans la déclaration de la syntaxe, quatre niveaux d'isolation sont disponibles :

- **READ UNCOMMITED** : Possibilité de lire des informations en cours d'insertion, mais non validées.
- **READ COMMITED** : Des données peuvent être modifiées avant la fin de la transaction.
- **REPEATABLE READ** : De nouvelles lignes peuvent apparaître avant la fin de la transaction.
- **SERIALIZABLE** : Les transactions sont gérées en série (valeur par défaut, normalement).

Les principaux types d'erreurs d'isolation à éviter sont les suivantes (le tableau 5 montre les effets des options de transactions sur celles-ci) :

- **La lecture impropre** : Lecture de données pas encore validées (e.g. surbooking lors d'annulation d'ajout de place pendant qu'un utilisateur prend des places).
- **La lecture non répétable** : Deux lectures successives ne donnent pas le même résultats (modification d'une valeur entre deux SELECT). Peut produire du surbooking, par exemple.
- **La lecture fantôme** : Des données apparaissent ou disparaissent dans des lectures successives (insertion de lignes entre deux sélections conditionnelles). Par exemple, nous pouvons obtenir un vol complet sans personne dedans.

La logique voudrait que l'option **SERIALIZABLE** soit toujours active. Mais celle-ci ralentit le serveur. Le mieux est encore de réfléchir aux actions que la base devra faire et choisir le type de transaction adaptée.

La Cohérence

La Cohérence est principalement faite manuellement, il faut faire attention, en autre, aux points suivant :

- Toute donnée référencée par une autre table ne doit pas être supprimée.
- Toute donnée référencée doit être supprimée avec toutes les lignes filles associées.

Erreur	READ UNCOMMITED	READ COMMITED	REPEATABLE READ	SERIALIZABLE
Lecture impropre	possible	impossible	impossible	impossible
Lecture non répétable	possible	possible	impossible	impossible
Lecture fantôme	possible	possible	possible	impossible

TABLE 5 – Niveaux d'isolation possibles et applications

– Toute modification d'une référence d'une donnée doit être répercutée dans toutes les lignes filles ou bien toute modification de cette référence doit être interdite si des lignes filles l'utilise.
– ...

Afin d'aider, il existe tout de même des outils comme les **TRIGGER** et les **modes de gestion d'intégrité** (les options dépendent du système de gestion de base de données utilisé) :

```
-- Mode de gestion des suppression (a définir lors de l'ajout de la clefs secondaire)
ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}

-- Mode de gestion des modification
ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}
```

Les options **NO ACTION** et **CASCADE** permettent, respectivement, d'annuler une modification de la clef (ou une suppression) s'il existe des référencements au données traitées ou d'exécuter l'action (en cascade) sur tous les éléments référençant les données.

Les options **SET DEFAULT** et **SET NULL** permettent d'effectuer la modification voulue (suppression ou modification de la clef) et d'éviter les erreurs en remplaçant les valeurs de référencement des lignes filles par la valeur par défaut ou bien par la valeur NULL.

Le deuxième outils permettant la gestion de cette cohérence est l'utilisation de **TRIGGER**, qui consiste en un morceau de code se déclenchant sous certaines conditions (que nous pouvons choisir). Il y a un exemple de **TRIGGER** en section 2.2. Pour rappel, sa syntaxe est :

```
CREATE TRIGGER nom_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nom_table
[FOR each row]
code
```

Les raisons d'utiliser un **TRIGGER** sont assez nombreuses :
– Le formatage de données lors de leur insertion (e.g. une date).
– Le remplissage automatique d'une clef.
– La suppression en cascade.
– L'abandon d'insertion si les données ne sont pas valables.
– L'abandon d'une suppression si elle entraine des lignes orphelines.

Note :
– Pour annuler une modification, nous pouvons utiliser la clause **ROLLBACK** dans un **TRIGGER**.
– Dans un **TRIGGER**, il existe deux pseudo-tables OLD et NEW qui contiennent les données avant et en cours de modification.
– L'option **FOR each row** permet d'exécuter le bout de code sur toute la table.

La durabilité

La durabilité implique que les données doivent être conservées, même si un problème arrive. Pour cela, nous pouvons :
– Sauvegarder la base après chaque transaction.
– Utiliser des disques en miroir.
– ...

2.6 Le SQL intégré

Il s'agit d'éléments SQL intègrés à un langage hôte (comme du PHP ou du C) pour pouvoir utiliser une base de données au sein d'une application ou d'un site Web, par exemple. Pour plus d'information sur l'utilisation du SQL au sein du PHP, voir la section 4.5.

3 PHP

De plus en plus de sites sont liées à des bases de données (e.g. gestion d'utilisateurs, flux d'informations...). Une bête page HTML statique ne peut pas être actualisée en fonction de ces données (e.g. changement d'utilisateur, nouvel article...). Un langage à vue le jour pour palier à ce problème : **PHP**. C'est un langage de scripts qui permet de générer des pages web dynamiquement : lorsqu'un nouvel utilisateur se connecte, un script est exécuté sur le serveur et génère une page HTML en fonction de ses données (e.g. afficher son nom sur une page). En outre, le PHP permet l'écriture de programme procédurale ou objet.

Le PHP est un langage communautaire et le meilleur moyen de trouver des informations est encore d'aller sur le site officiel : `http://www.php.net/manual/fr/`.

Pour de plus amples informations :
- `http://sylvie-vauthier.developpez.com/tutoriels/php/grand-debutant/`
- `http://g-rossolini.developpez.com/tutoriels/php/cours/`
- `http://www.lephpfacile.com/cours/`
- `http://g-rossolini.developpez.com/tutoriels/php/cours/`
- ...

3.1 Avant de commencer

Comme pour la plupart des langages, les fichiers PHP n'ont pas besoin d'être écrit à partir d'un logiciel de développement. Le simple bloc-note Windows (ou Unix) suffit. Un petit conseil tout de même, préférer Notepad++ (sous Windows, disponible à l'adresse `http://notepad-plus-plus.org/fr/`) qui permet d'avoir une coloration syntaxique du code (coloration des mots clefs). De plus, le PHP étant un code exécuté sur un serveur Web, il faut un serveur Web et un interpréteur PHP local pour pouvoir développer sur sa machine.

Pour créer un fichier contenant un script PHP, il faut lui associer une extension `.php` ainsi qu'utiliser les balise `<?php ... ?>`, à l'intérieur de celui-ci, pour indiquer quelles sont les parties codées en PHP. En effet, en dehors de ces balises, un autre langage peut être utilisé, comme du HTML. La balise d'ouverture est `<?php` et la balise de fermeture `?>`.

3.2 Élements syntaxiques et variables

La plupart des éléments syntaxiques du PHP viennent du langage C :
- Une instruction doit se terminer par un point virgule `;`
- Les commentaires sont introduits de deux manières :
  - Le commentaire sur une ligne : ce qui suit le symbole `//`
  - Le bloc de commentaire : ce qui est entre les deux symboles `/*` et `*/`
- Il existe les mêmes opérateurs (`==`, `!=`, `>`, `<`, ...).
- ...

A noter : Il existe d'autres opérateurs de comparaison entre types et classes : `===`, `!==`, `instanceof`.

Au sein de ce langage, les variables n'ont pas l'obligation d'être typées : il n'y a pas besoin de définir quelle sorte de données la variable va contenir. C'est un outils pratique, mais piégeur. De plus, cette liberté a un coût et entraine une syntaxe particulière sur les variables :
- Une variable doit commencer par $.
- Le caractère suivant $ doit être une lettre.

En plus des variables pouvant être "déclarées" par l'utilisateur, il en existe des globales présentent dans le langage (i.e. ces variables sont disponibles à tout moment, sans déclaration préalable) :
- `DIRECTORY_SEPARATOR` : C'est une chaîne de caractère définissant le séparateur d'adresse.
- `$_GET` : Le tableau de valeur provenant de l'URL.
- `$_POST` : Le tableau de valeur envoyé en méthode POST.
- `$_FILE` : Les données du fichier envoyées par formulaire.
- `$_SERVER` : Les valeurs misent en place par le serveur Web (elles peuvent donc changer d'une configuration à l'autre).
- `$_ENV` : Les variables d'environnement (système d'exploitation).
- `$_SESSION` : Les valeurs mises dans le magasin des sessions (stockées sur le serveur).
- `$_COOKIE` : Les valeurs transmises au moyen de cookies par le navigateur (stockés sur la machine cliente).
- `$GLOBALS` : L'ensemble des variables du script.

ATTENTION :
Ces variables peuvent produire des failles de sécurité. Il est préférable de vérifier toutes les données provenant de celle-ci avant de les utiliser.

En plus des variables classiques, il est aussi possible de déclarer des **constantes** :

```
<?php
        define("NOM_ADMIN", "Yogui");
        echo NOM_ADMIN; // On remarque l'absence de guillemets et de signe dollar
?>
```

Tout comme les variables globales, il existe des constantes native du langage qui ne nécessitent pas de déclaration, comme :
- `__LINE__` : La ligne de code en cours.
- `__FILE__` : Le nom complet du script en cours.
- `__DIR__` : Le nom du répertoire du script en cours (depuis les versions 5.3 et 6.0 de PHP).
- `__FUNCTION__` : La fonction en cours.
- `__CLASS__` : La classe en cours, similaire à `get_class($this)`.
- `__METHOD__` : La méthode en cours.
- `__NAMESPACE__` : L'espace de noms en cours (depuis les versions 5.3 et 6.0 de PHP).

Note : Généralement, les variables sont en minuscules et les constantes en majuscules.

Pour terminer avec les variables, voici une liste, non exhaustive, de fonctions utiles pour leurs gestions :
- `isset($variable)` : Détermine si une variable est définie et est différente de `NULL`.
- `empty($variable)` : Détermine si une variable est vide.
- `unset($variable)` : Détruit une variable.
- `defined(CONSTANTE)` : Vérifie l'existence d'une constante.
- `get_defined_constants()` : Retourne la liste des constantes et leurs valeurs.

– `constant(CONSTANTE)` : Retourne la valeur d'une constante.
– ...

3.3 Les tableaux

Les variables n'étant pas typées, la gestion des tableaux en est simplifiée. Un tableau peut être défini entièrement dès son premier appel ou bien s'agrandir lors de son utilisation. La fonction de création d'un tableau est `array()`. L'appel d'un élément du tableau est fait au moyen des crochets `[` et `]`. L'élément permettant d'accéder à une case du tableau est appelé **index** et peut être un nombre ou une chaîne de caractères (dans ce cas, **le tableau sera dit *associatif***). Si les index sont donnés en nombre, le premier élément du tableau est accessible via l'index 0.

Quelques fonctions existent pour la gestion de tableaux dont :
– `array()` : Créer un tableau.
– `count()` : Compter le nombre d'éléments d'un tableau.
– `sort()` : Trier un tableau.
– `list()` : Assigne des variables comme si elles étaient un tableau.

Voici un petit exemple d'utilisation de tableau :

```
<?php
        $tableau_fixe = array(3, 6, 9); //ce tableau contient trois valeurs : 3, 6, 9

        echo $tableau_fixe[0]."<br/>"; //accès direct à l'élément d'index 0
        print_r($tableau_fixe); //affichage du tableau complet

        foreach($tableau_fixe as $index => $valeur)
        {
                echo "<br/>[".$index."] => ".$valeur;
        }

        $tableau_dynamique = array(); //ce tableau est vide
        $tableau_dynamique["premier"] = 3 ;
        $tableau_dynamique["deux"]    = 6 ;
        $tableau_dynamique["toto"]    = 9 ;

        foreach($tableau_dynamique as $index => $valeur)
        {
                echo "<br/>[".$index."] => ".$valeur;
        }

        // Utilisation de list()
        list($element1,$element2,$element3) = $tableau_fixe ;
        echo "<br/>".$element1."<br/>".$element2."<br/>".$element3."<br/>" ;
?>
```

3.4 Les structures de contrôles

Les structures de contrôles, de bases, sont les mêmes qu'en C.

La conditionnelle `if`

```
if(<expression>) { <instruction> } else { <instruction> }
```

Exemple :

```
<?php
        $toto=-10;
        if ($toto>=0) {echo "$toto est positif";} else {echo "$toto est négatif";}?
?>
```

La condition ?

```
(<expression>)? <instruction si vrai> : <instruction si faux>
```

Exemple :

```
<?php
        $toto=10;
        echo ($toto>=0)? "$toto est positif":"$toto est négatif";
?>
```

La conditionnelle switch

```
switch(<expression>)
{
    case <valeur 1>:
        <instructions>
    break;

    case <valeur 2>:
        <instructions>
    break;

    ...

    default:
        <instructions>
}
```

Exemple :

```
<?php
        $toto="Visiteur";
        switch($toto)
        {
                case "Visiteur":
                        echo "Vous êtes un Visiteur !" ;
                break;

                case "Admin":
                        echo "Vous êtes un Admin !" ;
                break;

                default:
                        echo "Utilisateur non reconnu !" ;
        }
?>
```

La boucle for

```
for(<initialisation> ; <continuer tant que> ; <incrémentation>)
{
     <instructions>
}
```

Exemple :

```
<?php
        for($i=1;$i<10;$i++)
        {
                echo $i ;
        }
?>
```

La boucle `while`

```
while(<continuer tant que>)
{
     <instructions>
}
// Ou encore
do
{
     <instructions>
} while(<continuer tant que>);
```

Exemple :

```
<?php
        $i = 1 ;
        while($i<10)
        {
                echo $i ;
                $i++ ;
        }
?>
```

Il existe d'autres structures de contrôles facilitant l'**utilisation de tableaux** :

- La boucle `each` est prévue pour parcourir un tableau (en faisant avancer son pointeur interne à chaque appel). *Exemple :*

```
<?php
        $membres = array("BrYs", "mathieu", "Yogui");
        print_r(each($membres));
        echo "<br>";
        print_r(each($membres));
        echo "<br>";
        print_r(each($membres));
?>
```

- La boucle `foreach` est sensiblement équivalente à `for($i=0 ; $i<=count(<tableau>) ; $i++)`. *Exemple :*

```
foreach(<tableau> as <clef> => <element>)
{
        <instructions sachant que <element> = <tableau>[<clef>]>
}
// Ou encore
foreach(<tableau> as <element>)
{
        <instructions sachant que <element> = each(<tableau>)>
}
```

3.5 Les fonctions

La création d'une fonction se fait grâce à la syntaxe suivante :

```
function <nom de la fonction> (<noms des paramètres> [ [ = <valeur par défaut>], ...])
{
    <instructions>
    return <valeur>; // (optionnel)
}
```

De nombreuses fonctions sont déjà programmées et peuvent être utilisées. Parmi celle-ci, des fonctions de gestion de fichiers, d'affichage ainsi que de base de données.

Une option permet, depuis la version 5.3, de fixer un **espace de noms** : `namespace`. Cette fonctionnalité permet de définir des classes, fonctions... avec des noms identiques, dans des endroits différents (une application en est faites en section 5.1.4).

3.6 Les chaînes de caractères et l'affichage

L'affichage du contenu d'une variable, ou d'une chaîne de caractères, se fait généralement au moyen de la fonction `echo`. Pour le débogage des scripts, les fonctions `var_export()`, `print_r()` et `var_dump()` sont utiles :

- `echo <chaîne de caractère>` : Afficher une chaîne de caractères.
- `var_dump(<variable>)` : Affiche les informations de `<variable>`.
- `var_export(<variable>)` : Retourne le code PHP utilisé pour générer une variable.
- `print_r()` : Affiche des informations lisibles pour une variable.

Il est parfois utile de concaténer des chaînes de caractères, pour l'affichage ou bien une utilisation ultérieure. Une telle opération est simplifiée avec le PHP et sa mise en oeuvre est faites grâce au point.
Exemple :

```
<?php
        $mon_nom = "Toto" ;
        $texte = "Bonjour " ;
        $maintenant = date("l jS \of F Y h:i:s A") ;
        echo  $texte.$mon_nom.", nous sommes le ".$maintenant;
?>
```

Note : Il existe deux moyens de créer des chaînes de caractères : l'encadrement par des guillemets simples ' et doubles ". La différence principale vient du fait que la chaîne entre deux ' n'est pas interprétée, au contraire de ce qu'il y a entre deux ".

```
$var = 'test'; //Ou $var = "test"; ça ne change rien ici
echo 'la variable est $var'; // renvoie : la variable est $var
echo "la variable est $var"; // renvoie : la variable est test
```

3.7 La gestion des fichiers

Lire et écrire dans un fichier se fait de la même façon qu'en C :

1. Ouverture du fichier avec `fopen()` (en lecture ou écriture).
2. Lecture/Ecriture des données avec `fread()` ou `file_get_contents()` ou `readfile()`.
3. Fermeture du fichier avec `fclose()`.

Exemple :

```php
<?php
        $donnee_a_ecrire = "La lecture et l'écriture d'un fichier se fait comme en C :\r\n";
        $donnee_a_ecrire = $donnee_a_ecrire."\t-Ouverture du fichier\r\n";
        $donnee_a_ecrire = $donnee_a_ecrire."\t-Lecture/Ecriture des données\r\n";
        $donnee_a_ecrire = $donnee_a_ecrire."\t-Fermeture du fichier";

        // Ouverture d'un fichier en ecriture, ici, le pointeur se situe en debut de fichier
        $file_a_ecrire = fopen("file.txt", "w");
        // Ecriture
        $fwrite = fwrite($file_a_ecrire, $donnee_a_ecrire);
        // Fermeture
        fclose($file_a_ecrire) ;

        // Lit un fichier et l'affiche
        $nom_fichier = "file.txt";
        $file_a_ecrire = fopen($nom_fichier, "r");
        $donnees_brute = file_get_contents($nom_fichier);
        fclose($file_a_ecrire);
        echo "<br/><br/>Voici les données du fichiers :<br/>".$donnees_brute ;

        // Plus simple
        echo "<br/><br/>Deuxième méthode :<br/>";
        $donnees_tableau=file($nom_fichier);
        foreach($donnees_tableau as $ligne) echo "$ligne<br/>";
?>
```

3.8 La gestion des dates

PHP fournis quelques fonctions de gestion du temps. Nous en verrons cinq :
- `time`.
- `getdate`.
- `date`.
- `mktime`.
- `checkdate`.

La fonction `time(void)` retourne le *timestamp* UNIX actuel, elle fournit donc **la date courante**. Il s'agit d'un entier représentant le nombre de secondes écoulées depuis le 1er janvier 1970 00 :00 :00 GMT.

La fonction `getdate([ int \$timestamp = time() ] )` retourne un tableau associatif contenant **les informations de date et d'heure** du *timestamp* lorsqu'il est fourni, sinon, le *timestamp* de la date/heure courante locale. Le tableau retourné est défini par les clefs de la table 6.

Clé	Signification	valeurs possibles
"hours"	Heures au format numérique	0 à 23
"minutes"	Minutes au format numérique	0 à 59
"seconds"	Secondes au format numérique	0 à 59
"mday"	Jour du mois courant au format numérique	1 à 31
"wday"	Numéro du jour de la semaine courante	0 (Dim.) à 6 (Sam.)
"mon"	Numéro du mois actuel	1 à 12
"year"	Année (sur 4 chiffres)	1999, 2003
"yday"	Numérique du jour de l'année	0 à 365
"weekday"	Jour de la semaine (en anglais)	Sunday à Saturday
"month"	Mois courant (en anglais)	January à December

TABLE 6 – Clefs du tableau associatif de getdate()

La fonction `date ( string $format [, int $timestamp = time() ] )` retourne une **date sous forme d'une chaîne**, au format donné par le paramètre `$format`, fournie par le paramètre *timestamp* ou la date et l'heure courantes si aucun *timestamp* n'est fourni. En d'autres termes, le paramètre *timestamp* est optionnel et vaut par défaut la valeur de la fonction `time()`. Les paramètres de format sont définis dans la table 7.

Par exemple, `date("d/m/Y à H:i:s")` retourne la date actuelle sous la forme d'une chaîne de caractères, organisée de la manière suivante : "jj/mm/yyyy à HH :mm :ss".

Caractère de format	Signification	valeurs possibles
Jour	—	—
d	Jour du mois, sur deux chiffres (avec un zéro initial)	01 à 31
D	Jour de la semaine, en trois lettres (et en anglais)	Mon à Sun
j	Jour du mois sans les zéros initiaux	1 à 31
l ('L' minuscule)	Jour de la semaine	Sunday à Saturday
N	Représentation numérique ISO-8601 du jour de la semaine	1 (pour Lundi) à 7 (pour Dimanche)
S	Suffixe ordinal pour le jour du mois, en anglais	st, nd, rd ou th. Fonctionne bien avec j
w	Jour de la semaine au format numérique	0 (pour dimanche) à 6 (pour samedi)
z	Jour de l'année	0 à 365
Semaine	—	—
W	Numéro de semaine dans l'année ISO-8601	Exemple : 42 (la 42ème semaine de l'année)
Mois	—	—
F	Mois, textuel	January à December
m	Mois au format numérique, avec zéros initiaux	01 à 12
M	Mois, en trois lettres, en anglais	Jan à Dec
n	Mois sans les zéros initiaux	1 à 12
t	Nombre de jours dans le mois	28 à 31
Année	—	—
L	Est ce que l'année est bissextile	1 si bissextile, 0 sinon.
o	L'année ISO-8601	Exemples : 1999 ou 2003
Y	Année sur 4 chiffres	Exemples : 1999 ou 2003
y	Année sur 2 chiffres	Exemples : 99 ou 03
Heure	—	—
a	Ante meridiem et Post meridiem en minuscules	am ou pm
A	Ante meridiem et Post meridiem en majuscules	AM ou PM
B	Heure Internet Swatch	000 à 999
g	Heure, au format 12h, sans les zéros initiaux	1 à 12
G	Heure, au format 24h, sans les zéros initiaux	0 à 23
h	Heure, au format 12h, avec les zéros initiaux	01 à 12
H	Heure, au format 24h, avec les zéros initiaux	00 à 23
i	Minutes avec les zéros initiaux	00 à 59
s	Secondes, avec zéros initiaux	00 à 59
u	Microsecondes	Exemple : 654321
Fuseau horaire	—	—
e	L'identifiant du fuseau horaire	Exemples : UTC, GMT, Atlantic/Azores
I (i majuscule)	L'heure d'été est activée ou pas	1 si oui, 0 sinon.
O	Différence avec l'heure de Greenwich (GMT)	Exemple : +0200 (en heures)
P	Différence avec l'heure Greenwich (GMT)	Exemple : +02 :00 (heure :minutes)
T	Abréviation du fuseau horaire	Exemples : EST, MDT ...
Z	Décalage horaire en secondes	-43200 à 50400
Date et Heure complète	—	—
c	Date au format ISO 8601	2004-02-12T15 :19 :21+00 :00
r	Format de date	Exemple : Thu, 21 Dec 2000 16 :01 :07 +0200

TABLE 7 – Format de date

La fonction `checkdate ( int $month , int $day , int $year )` **vérifie la validité d'une date** formée par les arguments. Une date est considérée comme valide si chaque paramètre est défini correctement. Elle renvoie un booléen.

La fonction `mktime ([ int $hour = date("H") [, int $minute = date("i") [, int $second = date("s") [, int $month = date("n") [, int $day = date("j") [, int $year = date("Y") [, int $is_dst = -1 ]]]]]] )` **retourne un *timestamp* UNIX** correspondant aux arguments fournis. Ce *timestamp* est un entier long, contenant le nombre de secondes entre le début de l'époque UNIX (1er Janvier 1970 00 :00 :00 GMT) et le temps spécifié. Les arguments peuvent être omis, de droite à gauche, et tous les arguments manquants sont utilisés avec la valeur courante de l'heure et du jour. Le paramètre `$is_dst` est le paramètre d'heure d'hiver mais est obsolète depuis PHP 5.1.0.

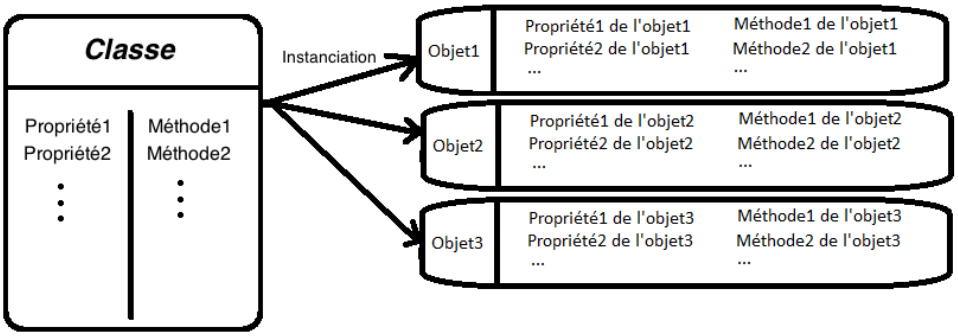

FIGURE 2 – Schéma de l'encapsulation

3.9 Le modèle objet

Dans la programmation structurée, les notions de variables et de fonctions sont séparées. La programmation objets permet de regrouper, dans un élément, des données et des fonctions associées : c'est **un objet**. Du point de vue du programme, il est définit par son nom, son état (via ses **propriétés** interne), et son comportement (via ses **méthodes** associées).

3.9.1 Définitions

Une classe est une représentation abstraite d'un objet : c'est le "moule" utilisé pour créer l'objet (e.g. une chaise, un livre, un humain, une rivière). Elle **définie *les propriétés* et les *méthodes* que l'objet contiendra.**

Une classe peut être rendue concrète au moyen d'**une instance de classe**, que l'on appelle aussi objet : on parle d'instancier la classe. L'objet ainsi créé à un état différent des autres : il est indépendant et a des propriétés qui lui sont propres (un schéma est donné en figure 2).

Un objet est donc un exemple concret de la définition abstraite qu'est la classe. Par exemple, on peut dire qu'un labrador croisé dans la rue est un objet de la classe chien.

Le modèle objet entraîne trois propriétés intéressantes :
- **L'héritage** : Créer une classe fille à partir d'une classe existante.
- **L'encapsulation** : Un objet contient toutes les propriétés (≈ variables) et les méthodes (≈ fonctions) permettant de se gérer en autonomie. De plus, certaines de ces propriétés et méthodes peuvent être cachés.
- **Le polymorphisme** : Définir, pour diverses classes, des méthodes de même nom (c'est le polymorphisme de *surcharge*) ou, pour des classes similaire (via héritage), de redéfinir des méthodes (c'est le *transtypage*).

PHP permet la gestion de ce modèle objets. Voici une liste de mots clefs permettant de déclarer des classes en PHP :
- `class` : Déclaration de classe.
- `const` : Déclaration de constante de classe.
- `function` : Déclaration d'une méthode.
- `public`/`protected`/`private` : Type d'accès (par défaut `public` si aucun accès n'est explicitement défini).
- `new` : Création d'objet.
- `self` : Appel à la classe elle-même.

- `parent` : Appel à la classe *parent*.
- `static` : Définition de variables statiques et appel statique (disponible depuis PHP 5.3 et 6.0).
- `$this` : Appel à l'objet en cours.
- `extends` : Héritage de classe.
- `implements` : Implémentation d'une *interface* (dont il faut redéclarer toutes les méthodes).

Une classe s'écrit au moyen du mot `class` suivi du nom de la classe et d'accolades (pas de point virgule). Les conventions d'écriture de code recommandent d'écrire ce mot comme un nom propre, à savoir avec une seule majuscule.

Au sein d'une déclaration de classe, il existe des variables qui permettent d'accéder à l'objet en cours (`$this`), à la classe en cours (`self`) ou encore, si la classe en cours est construite par héritage, au parent de celle-ci (`parent`). Ces variables sont utiles, par exemple, pour lire/modifier une variable de l'objet ou de la classe (variable statique ou constante).

L'accès à une méthode ou une propriété d'objet (s'il est public) se fait grâce à une flèche `->`. Il faut bien comprendre qu'**une méthode est une fonction appliquée à un objet** (son appel se fait via cette objet) : `<objet>-><methode>`.

Une propriété (ou méthode), déclarée comme `static` ou `const`, est commune à tous les objets d'une même classe. C'est une **propriété (ou méthode) de classe** puisqu'elle n'appartient pas à un objet en particulier. Le signe `::` permet son appel : `<objet>::<variable de classe>`.
Voici un exemple d'application d'une propriété `static` :

```
<?php
        class Caniche
        {
            // Propriétés
            public static $caniches = 0;

            // Méthodes
            public function __construct()
            // la méthode __construct est une méthode magique (cf. plus loin)
            {
                ++self::$caniches;
            }
        }

        // Utilisation de la classe
        $froufrou = new Caniche();
        $froufrette = new Caniche();

        echo $froufrou::$caniches; //affiche "2"
?>
```

Note : Les propriétés et méthodes des classes `parent`, `self` sont atteignables via à l'opérateur `::`

Voici un exemple de déclaration et d'instanciation de classe :

```
<?php
        //Création de la classe
        class Chien
        {
                /*****************************/
                /* Déclaration des propriétés */
                /*****************************/
                private $poids;
                private $taille;
                private $race;
                private static $nombre ; // ceci est une variable static
```

```
                /*****************************/
                /* Déclaration des méthodes   */
                /*****************************/

                // Le constructeur permet d'instancier un objet
                // Le contructeur doit porter le même nom que la classe
                // OU s'appeler __construct
                public function  Chien($poids=50,$taille=80,$race="inconnue")
                {
                        $this->poids=$poids;
                        $this->taille=$taille;
                        $this->race=$race;
                        self::$nombre++;
                }

                // La méthode afficher permet de connaître les détail de l'objet
                public function afficher()
                {
                        echo "Je suis un chien de race ".$this->race.
                                        ", je fais ".$this->poids."kg et ".$this->taille."cm.<br>";
                        echo "Il existe ".self::$nombre." chien(s) instancié(s).";
                }
        }

        // Instanciation d'un Chien
        $medor = new Chien();
        // Affichage des infos de médor
        $medor->afficher();
        echo "<br>" ;

        // Instanciation d'un nouveau chien : les proprietes sont differentes
        $pepette = new Chien(40,20,"doberman");
        // Affichage des infos de pepette
        $pepette->afficher()."<br/>";
?>
```

Note : Il existe des méthodes particulière, telle que le constructeur de classe, qui est une méthode permettant la création d'une nouvelle instance de classe. Le constructeur est défini par la méthode portant le nom de la classe ou bien par la méthode `__construct`, qui est une méthode magique (c.f. section 3.9.4).

Une interface est un ensemble de méthodes que les classes doivent définir si elles veulent l'implémenter. Elle permet la définition de méthodes publiques et la création d'un type général permettant l'appel d'un objet utilisant l'interface. Voici un exemple :

```
interface Joueur
{
    // une classe qui veut implémenter l'interface Joueur
    // doit définir la méthode jouer()
    public function jouer();
}

class Labrador implements Joueur
{
    public function jouer()
    {
        echo "Ouah!";
    }
}

$médor = new Labrador();
$médor->jouer();
```

3.9.2 L'héritage

L'héritage permet de définir une **hiérarchie de classes**. Par exemple, la classe `Chien`, précédemment définit, peut sembler trop abstraite : l'instanciation de `Chien` par un objet `$médor` ne permet pas de connaître sa race. Une possibilité serait d'ajouter une propriété `race` dans la classe `Chien`, mais la solution de l'héritage donne plus de souplesse : elle permet de redéfinir des propriétés et des méthodes de la classe `Chien` (grâce au **polymorphisme**), tout en héritant de celles non modifiées.

Certaines recommandations (PEAR, Zend Framework, etc.) préconisent d'utiliser l'underscore pour identifier l'héritage, en voici un exemple :

```
<?php
        // Création de la classe animal
        class Animal
        {/*Définition des propriétés et méthodes*/}

        // Création de sous-classe par héritage
        class Animal_Chien extends Animal
        {/*Définition des propriétés et méthodes, si besoin*/}
        class Animal_Chien_Caniche extends Animal_Chien
        {/*Définition des propriétés et méthodes, si besoin*/}
        class Animal_Chien_Labrador extends Animal_Chien
        {/*Définition des propriétés et méthodes, si besoin*/}
?>
```

3.9.3 L'encapsulation

L'encapsulation permet l'auto-gestion de tout objet instancié. Elle permet aussi de rendre publique, ou non, certaines propriétés et méthodes.

Pour gérer cette encapsulation, il existe trois mots clefs à utiliser au moment de la déclaration de la classe :

- `private` : Les propriétés et méthodes déclarées comme privées peuvent uniquement être vues et utilisées en interne et sont invisibles de l'extérieur.
- `protected` : Les propriétés et méthodes déclarées comme protégées peuvent uniquement être vues et utilisées en interne ou par d'autre objets de la même classe ou dérivés (par héritage) et sont invisibles pour des classes étrangères.
- `public` : Les propriétés et méthodes déclarées comme `public` sont vues et utilisées par tous les objets.

Pour avoir une gestion de la sécurité d'une propriété utilisable pas l'utilisateur, nous pouvons la rendre privée (en la déclarant `private`) et créer un **accesseur** et un **manipulateurs** (ou mutateur) pour, respectivement, la lire et la modifier. Ceci nous permet, par exemple, de vérifier les données saisies.

Exemple d'implantation d'une classe `Chien` ainsi que de classes filles, avec utilisation des mots clefs `self` et `parent` ainsi que d'un accesseur et un manipulateur :

```
<?php
        class Chien
        {
                private $age = 0 ; // Par defaut $age = 0

                protected function aboyer()
                {
                        return "Je suis un chien";
                }

                // Manipulateur de la propriété $age
                function set_age($nv_age)
```

```
                {
                        if (is_int($nv_age))
                                $this->age = $nv_age ;
                }
                // Accesseur de la propriété $age
                function get_age()
                {
                        return $this->age;
                }
        }
        class Chien_Labrador extends Chien
        {
                // Création d'une classe fille
                // On peut redéclarer les éléments publics ou protégés, mais pas ceux privés
                protected function aboyer()
                {
                        return "Je suis un labrador";
                }

                public function identifierParent()
                {
                        return parent::aboyer();
                }
                public function identifierSelf()
                {
                        return self::aboyer();
                }
        }

        $medor = new Chien_Labrador();
        echo $medor->identifierParent()."<br/>";
        echo $medor->identifierSelf()."<br/>";

        // Test de l'accesseur et du mutateur
        echo $medor->get_age()."<br/>" ;
        $medor->set_age("zut") ;
        $medor->set_age(1) ;
        $medor->set_age(0.1) ;
        echo $medor->get_age()."<br/>" ;
?>
```

Récemment, une nouvelle méthode d'accès a été mise en place, **le Late Static Bindings (LSB)**. Pour utiliser ce mode d'accès, il faut utiliser le préfixe `static::` devant une méthode. Voici un cas d'exemple (une brève explication est donnée par la suite) :

```
<?php
        class Chien
        {
            protected function aboyer()
            {
                return "Je suis un chien";
            }

            public function identifier_static()
            {
                return static::aboyer(); //appel statique
            }

            public function identifier_normal()
            {
                return self::aboyer(); //appel à la classe
            }
        }

        class Chien_Labrador extends Chien
        {
            protected function aboyer()
            {
                return "Je suis un labrador";
```

```
            }
        }

        $medor = new Chien();
        $felix = new Chien_Labrador();

        echo $medor->identifier_normal()."<br/>"; // Renvoie : Je suis un chien
        echo $medor->identifier_static()."<br/>"; // Renvoie : Je suis un chien
        echo $felix->identifier_normal()."<br/>"; // Renvoie : Je suis un chien
        echo $felix->identifier_static()."<br/>"; // Renvoie : Je suis un labrador
?>
```

Le LSB nous permet d'appeler la méthode lors de l'exécution, jusqu'alors l'interpréteur ne sait pas qu'elle sera la classe qui en fera l'appel. Alors que si nous utilisons le préfixe `self`, l'interpréteur considère que la seul classe pouvant utiliser la méthode `aboyer()` est `Chien`, puisque c'est dans celle-ci qu'elle a été déclarée.

Une fonction a fait son apparition avec le LSB : `get_called_class()`. Elle s'avère pratique lors de l'utilisation de base de données grâce à de la programmation objets, par exemple :

```
<?php
        class Table
        {
            static function getById($id)
            {
                return "SELECT * FROM ".get_called_class()." WHERE id = ".(int)$id;
            }
        }

        class Album  extends Table{}
        class Artist extends Table{}

        echo Album::getById(3); // SELECT * FROM Album WHERE id = 3
        echo Artist::getById(6); // SELECT * FROM Artist WHERE id = 6
?>
```

ATTENTION :
Cette **méthode d'accession** n'est pas à confondre avec les variables et méthodes `static` communes à l'ensemble des objets d'une même classe et accessible au moyen de l'opérateur `::`

3.9.4 Le polymorphisme

Le polymorphisme permet de définir, pour diverses classes, des méthodes de même nom. Nous pouvons en créer, ou utiliser des fonctions globales pour travailler sur n'importe qu'elle type d'objets. C'est le cas de la méthode permettant l'**instanciation de classe** (méthode `new`) ou encore celle du **du clonage** (méthode `clone`) qui est une méthode permettant de copier un objet entier (l'utilisation de l'opérateur "=" sur un objet ne permet que de recopier son pointeur – son adresse mémoire).

```
$obj_1 = new Test();
$obj_2 = clone $obj_1; // Creer un nouvel objet identique à $obj_1
```

Bien entendu, ces méthodes peuvent être changées au besoin (comme dans un des exemples précédent où le constructeur avait été redéfini pour compter le nombre d'objet, via `__construct()`).

Les méthodes permettant de changer l'action des fonctions de traitement préexistantes sont des **méthodes magiques**. En voici une liste :

- `__construct()` : Le constructeur de la classe, qui est exécuté à la création d'un nouvel objet.
- `__destruct()` : Le destructeur de la classe, qui est exécuté à la suppression d'un objet.
- `__set()` : Déclenchée lors de l'accès en écriture à une propriété de l'objet.
- `__get()` : Déclenchée lors de l'accès en lecture à une propriété de l'objet.

- `__call()` : Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel non statique).
- `__callstatic()` : Déclenchée lors de l'appel d'une méthode inexistante de la classe (appel statique) : disponible depuis PHP 5.3 et 6.0.
- `__isset()` : Déclenchée si on applique `isset()` à une propriété de l'objet.
- `__unset()` : Déclenchée si on applique `unset()` à une propriété de l'objet.
- `__sleep()` : Exécutée si la fonction `serialize()` est appliquée à l'objet (elle permet de linéariser une variable).
- `__wakeup()` : Exécutée si la fonction `unserialize()` est appliquée à l'objet (elle effectue l'opération inverse de `serialize()`).
- `__toString()` : Appelée lorsque l'on essaie d'afficher directement l'objet : `echo $object`.
- `__set_state()` : Méthode statique lancée lorsque l'on applique la fonction `var_export()` à l'objet.
- `__clone()` : Appelé lorsque l'on essaie de cloner l'objet.
- `__autoload()` : Cette fonction n'est pas une méthode, elle est déclarée dans le scope global et permet d'automatiser les `include`/`require` de classes PHP.

Note : Dans l'exemple de l'accesseur et du manipulateur, nous aurions aussi pu utiliser les deux méthode magique `__set()` et `__get()` (qui se déclenchent pour des propriétés non accessibles). *Exemple :*

```
<?php
        class Chien
        {
                // Propriétés
                private $age =0  ; // Par defaut $age = 0
                private $notes  ;

                // Manipulateur
                function __set($propertie_name,$value)
                {
                        switch ($propertie_name)
                        {
                                case "age":
                                        if (is_int($value))
                                                $this->age = $value ;
                                break;

                                default:
                                        $this->$propertie_name = $value ;
                        }
                }

                // Accesseur
                function __get($propertie_name)
                {
                        return $this->$propertie_name;
                }
        }
        $medor = new Chien();
        // Test de l'accesseur et du mutateur
        $medor->notes="RAS" ;
        echo $medor->notes."<br/>" ;
        echo $medor->age."<br/>" ;
        $medor->age="zut" ;
        $medor->age=1 ;
        $medor->age=0.1 ;
        echo $medor->age."<br/>" ;
?>
```

Evidemment ces **méthodes magiques** ne sont pas les seules outils pré-implantés dans le langage, il existe beaucoup de fonctions et de constantes pouvant être utiles lors de la création et l'utilisation des classes :

– Fonctions utiles sur les classes :
  • `class_parents()` : Retourne un tableau de la classe parent et de tous ses parents.
  • `class_implements()` : Retourne un tableau de toutes les interfaces implémentées par la classe et par tous ses parents.
  • `get_class()` : Retourne la classe de l'objet passé en paramètre.
  • `get_called_class()` : À utiliser dans une classe, retourne la classe appelée explicitement dans le code PHP et non au sein de la classe.
  • `class_exists()` : Vérifie qu'une classe a été définie.
  • `get_class()` : Retourne la classe d'un objet.
  • `get_declared_classes()` : Liste des classes définies.
  • `get_class_methods()` : Liste des méthodes d'une classe.
  • `get_class_vars()` : Liste des propriétés d'une classe.
– Constantes utiles sur les classes :
  • `__CLASS__` : Donne le nom de la classe en cours.
  • `__METHOD__` : Donne le nom de la méthode en cours.

Note : La fonction `get_called_class()` s'apparente à la constante magique `__CLASS__`, à la différence qu'elle retourne la classe appelée par le développeur plutôt que la classe actuelle dans le code.

Enfin, il existe aussi des classes pré-implantées. Nous n'en citerons qu'une ici, **la classe Exception**. Elle permet de gérer des cas particuliers. Un exemple est l'indisponibilité du serveur pendant la connexion à une base de données : ce n'est pas une situation normale, néanmoins elle est facilement prévisible et identifiable. Voici la définition de cette classe :

```
Exception {
        /* Properties */
        // The exception message
        protected string $Exception->message ;
        // The Exception code
        protected int $code ;
        // The filename where the exception was created
        protected string $file ;
        // The line where the exception was created
        protected int $line ;
        /* Methods */
        // Construct the exception
        public Exception::__construct ([ string $message = "" [, int $code = 0
                                                                [, Exception $previous = NULL ]]] )
        // Gets the Exception message
        final public string Exception::getMessage ( void )
        // Returns previous Exception
        final public Exception Exception::getPrevious ( void )
        // Gets the Exception code
        final public mixed Exception::getCode ( void )
        // Gets the file in which the exception occurred
        final public string Exception::getFile ( void )
        // Gets the line in which the exception occurred
        final public int Exception::getLine ( void )
        // Gets the stack trace
        final public array Exception::getTrace ( void )
        // Gets the stack trace as a string
        final public string Exception::getTraceAsString ( void )
        // String representation of the exception
        public string Exception::__toString ( void )
        // Clone the exception
        final private void Exception::__clone ( void )
}
```

Voici un exemple de lancement et d'identification d'une exception :

```
function inverse($x)
{
    if (!$x)
    {
        throw new Exception("Division par zéro.");
    }
    else return 1/$x;
}

try
{
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
}
catch (Exception $e)
{
    echo "Exception reçue : ",  $e->getMessage(), "\n";
}

// Continue execution
echo "Bonjour le monde !";
```

Note :
- La **création** d'une instance de classe se fait au moyen du mot clef `new`.
- L'**accès** aux propriétés et méthodes de l'objet se fait par la flèche `->`, SAUF pour les propriétés et méthodes `static` qui sont atteignable par l'opérateur `::`
- Les propriétés et méthodes des classes `parent`, `self` sont atteignables via à l'opérateur `::`
- Le point n'est pas utilisé comme dans la POO de C/C++ ou Flash.

4 La construction d'une page dynamique

Maintenant connues les bases de programmation en PHP, nous pouvons commencer à utiliser ce langage pour créer des pages Web dynamiques. Nous verrons quatre exemples d'applications : l'utilisations des formulaires, l'upload de fichiers, la gestion des cookies et l'interaction avec une base de données.

4.1 Les formulaires

Un formulaire est constitué de balises HTML permettant d'interagir avec l'utilisateur. Pour rappel, un formulaire est contenue dans une balise `<FORM METHOD="POST"|"GET" [ACTION="type de fichier MIME"] [ENCTYPE = "type de cryptage"]> ... </FORM>`. Dans cette balise, plusieurs éléments peuvent y être contenus pour interagir avec l'utilisateur. Les données entrées seront envoyées à l'url (indiquée dans l'option `ACTION`) par la méthode fournie à l'attribut `METHOD`. Les balises interactives pouvant être contenues dans une balise `FORM` sont du type :
- `INPUT` : Un ensemble de boutons et de champs de saisie.
- `TEXTAREA` : Une zone de saisie.
- `SELECT` : Une liste à choix multiples.

La balise `<INPUT type="Nom du type" value="Valeur par défaut" name="Nom de l'élément">` peut avoir plusieurs types différents :
- `checkbox` : Il s'agit de **cases à cocher** pouvant admettre deux états : `checked` (cochée) et `unchecked` (non cochée).
- `hidden` : Il s'agit d'un **champ caché**.

- `file` : Il s'agit d'un champ permettant à l'utilisateur de préciser l'emplacement d'un **fichier qui sera envoyé** avec le formulaire. Il faut, dans ce cas, préciser le type de données pouvant être envoyées grâce à l'attribut `ENCTYPE` de la balise `FORM` (c.f. section 4.2).
- `image` : Il s'agit d'un **bouton de soumission personnalisé**, dont l'apparence est l'image située à l'emplacement précisé par son attribut `SRC`.
- `password` : Il s'agit d'un **champ de saisie**, dans lequel les caractères saisis apparaissent sous forme d'astérisques afin de camoufler la saisie de l'utilisateur.
- `radio` : Il s'agit d'un **bouton** permettant un choix parmi plusieurs proposés (l'ensemble des boutons radios devant porter le même attribut `name`). Un attribut `checked` pour un des boutons permet de préciser le bouton sélectionné par défaut.
- `reset` : Il s'agit d'un **bouton de remise à zéro** permettant uniquement de rétablir l'ensemble des éléments du formulaire à leurs valeurs par défaut.
- `submit` : Il s'agit du **bouton de soumission** permettant l'envoi du formulaire. Le texte du bouton peut être précisé grâce à l'attribut `value`.
- `text` : Il s'agit d'un **champ de saisie** permettant la saisie d'une ligne de texte. La taille du champ peut être définie à l'aide de l'attribut `size` et la taille maximale du texte saisi grâce à l'attribut `maxlength`.

Nous ne donnerons que les principales caractéristiques de la balise `TEXTAREA`. Pour plus d'informations, ce référer à `http://dev.w3.org/html5/markup/textarea.html`.

- `cols` : Représente le nombre de caractères que peut contenir une ligne.
- `rows` : Représente le nombre de lignes.
- `name` : Représente le nom associé au champ.
- `readonly` : Permet d'empêcher l'utilisateur de modifier le texte entré par défaut dans le champ.
- `value` : Représente la valeur par défaut.

La balise `SELECT` permet la création de **liste déroulante** d'éléments (chaque élément est contenu dans une balise `OPTION`). On peut ajouter l'option `multiple` si l'utilisateur est autorisé à sélectionner plusieurs lignes ou l'option `disabled` si l'on veut créer une liste non modifiable par l'utilisateur. L'élément marqué par `selected` sera l'élément sélectionné par défaut. Voici une syntaxe possible :

```
<SELECT name=<nom du champ> size=<nombre de ligne a afficher> >
        <option value=<valeur de la ligne> >valeur de l'élément 1
        <option value=<valeur de la ligne> selected>valeur de l'élément 2
        ...
</SELECT>
```

Note :

- Lors de la création du formulaire, la page Web peut **se référencer elle-même** grâce au PHP, et notamment à la variable d'environnement `$_SERVER["PHP_SELF"]`.
- La fonction PHP `isset()` peut s'avérer très utile pour tester la présence d'une valeur de formulaire dans les variables d'environnement telles `$_GET`, `$_POST` et `$_COOKIES`.
- Les fonctions `serialize()` (qui linéarise n'importe qu'elle variable PHP) et `unserialize()` (qui effectue la transformation inverse) peuvent s'avérer pratique si l'on souhaite **transmettre des objets PHP via un formulaire**.

Voici un petit exemple de formulaire :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
                <title> Un petit formulaire </title>
        </head>
        <body>
                <h2> Un formulaire...</h2>
                        <form method="GET" action=<?php echo $_SERVER["PHP_SELF"] ?>>
                                Quel est votre nom ?<br>
                                <input type="text" name="nom"><br>
                                <input type="submit" name="OK" value="Valider"><br>
                        </form>
                <!-- <script type="text/javascript">alert('bonjour')</script> -->
                <?php
                        if (isset($_GET["nom"]))
                        {
                                echo "<h2> Réponse du formulaire </h2>";
                                $today = getdate();
                                echo "<p>Salut, ".$_GET["nom"]."</p>
                                        <p>Nous sommes le ".$today["mday"].
                                                " ".$today["month"]." ".$today["year"].".</p>
                                        <p>Il est ".$today["hours"]." h ".$today["minutes"].
                                                " min ".$today["seconds"]." sec.</p>";
                        }
                ?>
        </body>
</html>
```

4.2 L'upload de fichier

L'upload de fichier permet, à un utilisateur, d'envoyer un fichier lui appartenant sur le Web. Afin d'effectuer une telle opération, nous pouvons utiliser un formulaire en `POST` associé à l'`ENCTYPE` `multipart/form-data` qui permet de spécifier qu'elle est l'encodage à utiliser pour envoyer les données.

L'interface utilisateur est faite simplement grâce à une balise `INPUT` de type `file`, à laquelle il faut associer un nom.

Une petite astuce permet de spécifier une taille de fichier maximale (en octet) dans une deuxième balise cachée, qui ressemblera à `<input type="hidden" name="MAX_FILE_SIZE" value="100000">`. Cette balise n'est, en soit, pas nécessaire puisque l'utilisateur peut la tromper (c.f. section 6) et que l'option `upload_max_filesize` de PHP permet de définir une taille de fichier maximal. Cependant, il est préférable de l'inclure pour informer l'utilisateur que son fichier est trop lourd, avant la fin de son téléchargement sur le serveur.

L'envoie du fichier étant possible grâce à un formulaire (en supposant que l'option PHP `file_uploads` est activée), il faut maintenant gérer les données côté serveur et donc utiliser PHP. Pour cela, il existe une variable d'environnement `$_FILES` qui donne accès aux propriétés du fichier téléchargé :

- Le nom : `$_FILES[<nom_du_fichier>]["name"]`
- Le chemin du fichier temporaire : `$_FILES[<nom_du_fichier>]["tmp_name"]`
- La taille (peu fiable, dépend du navigateur) : `$_FILES[<nom_du_fichier>]["size"]`
- Le type MIME (peu fiable, dépend du navigateur) : `$_FILES[<nom_du_fichier>]["type"]`
- Un code d'erreur si besoin : `$_FILES[<nom_du_fichier>]["error"]`

Cette variable d'environnement nous permet de gérer le fichier venant d'arriver sur le serveur, dans le dossier temporaire d'upload (choisi via les configurations de PHP, par `upload_tmp_dir`). Il faut maintenant le déplacer vers l'endroit voulu (e.g. un dossier `upload` dans la racine du serveur), cette action est possible grâce à la fonction `move_uploaded_file( string $filename , string $destination )` qui retourne `TRUE` si le déplacement a été effectué, `FALSE` sinon. Il est important que le serveur ait accès en écriture sur le dossier cible, cette action est faisable grâce à la fonction `chmod( string $filename, int $mode )` (c.f. `http://php.net/manual/fr/function.chmod.php`).

Pour résumer, afin d'uploader un fichier, il faut :

1. Construire une interface utilisateur grâce à un formulaire HTML.
2. Récupérer les informations du fichier arrivant sur le serveur grâce à la variable `$_FILES` de PHP.
3. Déplacer le fichier à l'endroit voulu sur le serveur via la fonction `move_uploaded_file()`.

Il est possible de mettre en place un upload de fichier avec ces fonctions. Cependant ce serait prendre de gros risques. En effet, rien n'empêche un utilisateur de mettre, sur le serveur, un fichier PHP pour récupérer le contenu d'une base de données ou pour effectuer une redirection vers un site plus que douteux...

Pour empêcher cela, nous pouvons ajouter une série de tests sur le fichier, avant son déplacement, comme une vérification d'extensions ou un formatage du nom (grâce à des fonctions de gestion des chaînes de caractères).

Voici un exemple d'application :

```
<html>

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title> Plop PHP </title>
  </head>

  <body>

    <form method="POST" action="fichier.php" enctype="multipart/form-data">
      <!-- On limite le fichier à 100Ko -->
      <input type="hidden" name="MAX_FILE_SIZE" value="1000000">
      Fichier : <input type="file" name="monFichier">
      <input type="submit" name="envoyer" value="Envoyer le fichier">
    </form>

    <?php
      if(isset($_FILES["monFichier"]))
      {
        $dossier = "upload/";
        $extensions_okay = array(".png", ".gif", ".jpg", ".jpeg");

        echo $_FILES["monFichier"]["name"]."<br/>";
        echo $_FILES["monFichier"]["tmp_name"]."<br/>";
        echo $_FILES["monFichier"]["type"]."<br/>";
        echo $_FILES["monFichier"]["size"]."<br/>";
        echo filesize($_FILES["monFichier"]["tmp_name"])."<br/>";

        $fichier = basename($_FILES["monFichier"]["name"]); // extrtait le nom du fichier
        $extension = strrchr($_FILES["monFichier"]["name"], "."); // extrait l'extension

        //Début des vérifications de sécurité...
        if(!in_array($extension, $extensions_okay)) //Si l'extension n'est pas dans le tableau
          $erreur = "Vous devez uploader un fichier de type png, gif, jpg, jpeg.";
        if($_FILES["monFichier"]["error"]>0)
          $erreur = "Erreur lors de l'upload.";
        if($_FILES["monFichier"]["error"]==2 || $_FILES["monFichier"]["error"]==1)
        {
          // error = 1. Le fichier téléchargé excède la taille de upload_max_filesize,
          //  configurée dans le php.ini.
          // error = 2. Le fichier téléchargé excède la taille de MAX_FILE_SIZE,
          //  qui a été spécifiée dans le formulaire HTML.
          $erreur = "Le fichier est trop gros...";
        }
        if (!is_uploaded_file($_FILES["monFichier"]["tmp_name"]))
          $erreur = "Attaque possible par téléchargement de fichier.";

        if(!isset($erreur)) //S'il n'y a pas d'erreur, on upload
```

```
        {
          //On formate le nom du fichier ici...
          $fichier = strtr($fichier,
              "ÀÁÂÃÄÅÇÈÉÊËÌÍÎÏÒÓÔÕÖÙÚÛÜÝàáâãäåçèéêëìíîïðòóôõöùúûüýÿ",
              "AAAAAACEEEEIIIIOOOOOUUUUYaaaaaaceeeeiiiioooooouuuuyy");
          $fichier = preg_replace("/([^.a-z0-9]+)/i", "-", $fichier); // Contient une expression régulière
          if(move_uploaded_file($_FILES["monFichier"]["tmp_name"], $dossier.$fichier))
          //Si la fonction renvoie TRUE, c'est que ça a fonctionné...
          {
            echo "Upload effectué avec succès !";
          }
          else
          //Sinon (la fonction renvoie FALSE).
          {
            echo "Echec de l'upload !<br>"."Code erreur ".$_FILES["monFichier"]["error"];
          }
        }
        else
        {
          echo $erreur;
        }
      }
    ?>
  </body>
</html>
```

4.3 Les en-têtes HTTP

Lors de chaque échange par le protocole HTTP (entre le navigateur de l'utilisateur et le serveur), des informations sur les données à envoyer (dans le cas d'une requête) ou envoyées (dans le cas d'une réponse) sont transmises dans des en-têtes HTTP. Ces en-têtes permettent aussi d'effectuer des actions sur le navigateur comme le transfert de cookie ou bien une redirection vers une autre page.

Ces en-têtes sont les premières informations envoyées au navigateur (pour une réponse) ou au serveur (dans le cas d'une requête), elles ont une syntaxe particulière de la forme `nom_en-tête: valeur` (l'espace n'est pas facultatif). Grâce à PHP, nous pouvons envoyer des en-têtes HTML facilement, en utilisant la fonction définie par : `booléen header(chaîne <en-tête HTTP>)`. Mais, comme les en-têtes sont les premières informations envoyées, cette fonction doit être appelée avant tout envoie de données HTML (i.e. l'utilisation de la fonction doit se situer avant la balise `<HTML>` ou toutes fonctions d'écriture, comme `echo()`).

La fonction `header()` peut, par exemple, être utilisée pour effectuer une redirection vers une nouvelle page ou encore faire un compteur statistique.

PHP fournit également une fonction permettant de récupérer un tableau associatif contenant l'ensemble des en-têtes HTTP envoyées par le navigateur, c'est la fonction `getallheaders()`.

Voici un exemple d'utilisation de header :

```
// Voici un premier fichier
<?php
  $header = getallheaders() ;
  if (!isset($header["location"]))
     header("Location: exemple_header2.php");
?>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title> Un petit formulaire </title>
  </head>
  <body>
    <h2> Un formulaire...</h2>
    <form method="GET" action=<?php echo $_SERVER["PHP_SELF"] ?>>
```

```
      Quel est votre nom ?<br>
      <input type="text" name="nom"><br>
      <input type="submit" name="OK" value="Valider"><br>
    </form>
  </body>
</html>

// Voici le deuxième fichier à nommer "exemple_header2.php"
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title> Un petit formulaire </title>
  </head>
  <body>
    <h2> Vous avez été redirigé......</h2>
    <?php
      foreach (getallheaders() as $name => $value)
      {
        echo "$name: $value\<br>";
      }
    ?>
  </body>
</html>
```

4.4 Les cookies

Un cookie est un fichier texte stocké sur le disque dur du client, permettant de mémoriser des informations.

PHP fournit une fonction permettant d'envoyer une demande de création de cookie sur une machine, c'est la fonction :

```
setcookie ( string \$name [, string \$value
            [, int \$expire = 0 [, string \$path
            [, string \$domain [, bool \$secure = false
            [, bool \$httponly = false ]]]]]
          )
```

Elle définit une demande de création de cookie, qui sera envoyé avec le reste des en-têtes. Comme pour les autres en-têtes, **les cookies doivent être envoyés avant toute autre sortie** (c'est une restriction du protocole HTTP, pas de PHP). Cela nous impose d'appeler cette fonction avant toute balise `<html>` ou `<head>` ou toutes autres fonctions d'affichage. Les paramètres sont les suivants :

- `$name` : Le nom du cookie.
- `$value` : La valeur du cookie. Cette valeur est stockée sur l'ordinateur du client, les informations importantes ne doivent pas y être stockées. Si le paramètre `$name` vaut `"cookiename"`, alors cette valeur est retrouvée en utilisant `$_COOKIE["cookiename"]`.
- `$expire` : Le temps après lequel le cookie expire. C'est un *timestamp* Unix, donc, ce sera un nombre de secondes depuis l'époque Unix (1 Janvier 1970). En d'autres termes, cette valeur peut être fixée à l'aide de la fonction `time()` en y ajoutant le nombre de secondes après lesquelles nous voulons que le cookie expire (e.g. `time()+60*60*24*30` fera expirer le cookie dans 30 jours). La fonction `mktime()` peut aussi être utilisée. Si ce paramètre n'est pas indiqué ou s'il vaut 0, le cookie expirera à la fin de la session (lorsque le navigateur sera fermé).
- `$path` : Le chemin, sur le serveur, où le cookie sera disponible. Si la valeur est `"/"`, le cookie sera disponible sur l'ensemble du domaine `$domain`. Si la valeur est `"/src/"`, le cookie sera uniquement disponible dans le répertoire `/src/` ainsi que tous ses sous-répertoires. La valeur par défaut est le répertoire courant où le cookie a été défini.
- `$domain` : Le domaine où le cookie est disponible. Pour rendre le cookie disponible sur tous les sous-domaines de `example.com`, il faut mettre la valeur `".example.com"`. Le point `"."`

n'est pas requis mais est nécessaire pour la compatibilité avec certains navigateurs. Indiquer `www.example.com` et le cookie sera disponible uniquement sur le sous-domaine `www.`
- `$secure` : Indique si le cookie doit uniquement être transmis à travers une connexion sécurisée HTTPS depuis le client. Lorsqu'il est positionné à `TRUE`, le cookie ne sera envoyé que si la connexion est sécurisée. Côté serveur, c'est au développeur d'envoyer ce genre de cookie uniquement sur les connexions sécurisées (e.g. en utilisant la variable `$_SERVER["HTTPS"]`).
- `$httponly` : Lorsque ce paramètre vaut `TRUE`, le cookie ne sera accessible que par le protocole HTTP. Cela signifie que le cookie ne sera pas accessible via des langages de scripts, comme Javascript. Il a été accepté que cette configuration (bien qu'elle ne soit pas supportée par tous les navigateurs) permet de limiter les attaques via XSS (c.f. section 6.2), d'autres ne sont pas de cet avis.

Voici un exemple d'utilisation de cookies :

```
<?php
  $value = "Valeur de test";
  setcookie("TestCookie", $value, time()+10);  /* expire dans 10 secondes */
?>

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title> Les cookies </title>
  </head>
  <body>
    <h2> Les cookies...</h2>
    <?php
      // Afficher un cookie
      if (isset($_COOKIE["TestCookie"]))
        echo $_COOKIE["TestCookie"]."<br>";
      else
        echo "Le cookie n'existe pas, rafraichir la page.<br>";

      // Une autre méthode pour afficher tous les cookies
      print_r($_COOKIE);
    ?>
  </body>
</html>
```

Note :
- Un cookie est utilisable par toutes les pages PHP contenues dans le chemin `path` (e.g. si le chemin est `"/"`, toutes les pages du site auront accès aux données du cookie).
- La **suppression du cookie** se fait en envoyant un cookie, du même nom, dont la date d'expiration est passée, `setcookie($name,"",1,"/","",0)`, ou bien en recréant un cookie vide, du même nom, `setcookie($name)`.
- Les cookies servent à stocker des informations chez l'utilisateur. Si nous souhaiterions stocker des informations sur le serveur (e.g. pour stocker un panier d'achat d'un utilisateur, entre deux connexions), nous pouvons utiliser une **session**. Chaque session est désignée par un nom et un identifiant. Chaque visiteur accédant à une page web se voit assigner un identifiant unique, appelé **identifiant de session**. Il peut être stocké soit dans un cookie, soit propagé dans l'URL.
- Pour plus de détails sur les cookies et les sessions, voir : `http://fr.php.net/manual/fr/book.session.php` ; `http://fr.php.net/manual/fr/features.cookies.php`

4.5 L'interaction avec une base de données

PHP ne pouvant pas stocker beaucoup de données sur le long terme, nous pouvons utiliser une base de données pour le faire. L'accès à cette base de données doit être invisible pour l'utilisateur,

d'où l'utilisation de PHP (qui va permettre de gérer les requêtes SQL). Cet accès, tout comme pour un fichier, est soumis à divers étapes :

1. Ouverture de la connexion au serveur.
2. Choix de la BdD sur le serveur (cela se fait parfois au moment de l'ouverture de la connexion).
3. Préparation des requêtes.
4. Exécution des requêtes et récupération des résultats.
5. Fermeture de la connexion.

Nous verrons deux façons de procéder. La première, étant spécifique à MySQL, n'est pas utilisable si le serveur final tourne avec un autre SGDB. La deuxième méthode est basée sur de la programmation orientée objets et offre donc plus de possibilités.

Note : Il y a deux avantages majeurs à préparer les requêtes :
- Chaque exécution de requête est plus rapide lorsqu'elle est préparée que lorsqu'elle est envoyée, en totalité, à chaque fois.
- Les attaques par injection SQL peuvent être réduites tout en éliminant le besoin de protéger les paramètres manuellement.

Note : Dans cette partie, nous avons besoin de l'utilisateur ROBERT (avec le mot de passe `lemotdepassederobert`) ayant les droits de SELECT, INSERT et DELETE sur une base nommée `essai` contenant les tables de la section 2.1 (hébergée en local). La création de l'utilisateur peut être faite avec PHPMyAdmin ou par le script SQL (c.f. section 2.4) :

```
GRANT SELECT,INSERT,DELETE
ON essai.*
TO ROBERT@127.0.0.1 IDENTIFIED BY "lemotdepassederobert"
```

Utilisation de l'API MySQL

L'utilisation de cette API est fortement déconseillée... Elle est présentée ici à titre d'exemple.

PHP fournit des fonctions de gestion de base de données pour un serveur utilisant MySQL (notre cas). L'utilisation d'une base de données, grâce à cette librairie, peut se faire avec les fonctions :
- `mysql_connect(<adresse du serveur>, <nom utilisateur>, <mot de passe>)` : Permet d'**ouvrir une connexion** avec le serveur. Renvoie un identifiant de connexion.
- `mysql_select_db (<base a selectionner> [, <identifiant de la connexion>])` : Permet de **sélectionner une base de données** du serveur. Si l'identifiant de connexion n'est pas fourni, la dernière connexion ouverte sera prise en compte.
- `mysql_query (<requete a envoyer> [, <identifiant de la connexion>])` : Permet d'**envoyer une requête**. Retourne une ressource, si c'est une demande d'informations (e.g. SELECT, SHOW), ou `TRUE`, si c'est une demande de modification (e.g. INSERT, DROP), ou `FALSE`, si la requêtes échoue. Si l'identifiant de connexion n'est pas fournis, la dernière connexion ouverte sera prise en compte.
- `mysql_fetch_assoc(<resource de mysql_query>)` : Retourne un tableau associatif qui contient la ligne lue dans `<resource de mysql_query>` et déplace le pointeur interne de données.
- `mysql_fetch_array(<resource de mysql_query> [, <type de resultat> ] )` : **Convertit une ressource** résultant de `mysql_query` en un tableau associatif ou un tableau indexé, ou les deux, en fonction de l'option `<type de resultat>`. Par défaut, la fonction renvoie les deux types de tableaux.

- `mysql_num_rows(<resource de mysql_query> )` : Retourne le **nombre de lignes d'un résultat** de `mysql_query`.
- `mysql_free_result(<resource de mysql_query> )` : Libère le résultat de la mémoire.
- `mysql_close ([<identifiant de la connexion>] )` : **Ferme la connexion** avec le serveur. Si l'identifiant de connexion n'est pas fournis, la dernière connexion ouverte sera prise en compte.

Voici un exemple d'application :

```
// Code à mettre dans un fichier "fonctions.php"
<?php
  function ConnexionBdd()
  // Fonction de connexion et sélection de la base de données
  {
    // Attention il peut y avoir des problèmes avec "localhost" sous Seven
    $base = mysql_connect ("127.0.0.1", "ROBERT", "lemotdepassederobert");
    mysql_select_db ("essai", $base) ; // On selectionne la base "essai"
  }
?>

// Code à mettre dans un fichier "mysql.php"
<?php
  // On importe les fonctions de gestion de la bdd
  include("fonctions.php");
?>

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Test connexion avec MySQL</title>
</head>
  <body>
    <?php
      // Premier script : on lit les informations
      // Connexion à la base
      ConnexionBdd();
      // Préparation de la requête
      // En pratique, le SELECT * est à éviter...
      $sql = "SELECT * FROM TABLE_ETUDIANT WHERE Section=\"SRC\"";
      // Envoie de la requête et arrêt du script s'il y a une erreur
      $db_result = mysql_query($sql) or die ("Erreur SQL !".$sql."<br />".mysql_error());

      // Lecture du résultat ligne par ligne
      while($row = mysql_fetch_assoc($db_result))
      {
        print_r($row);
        echo "<br>";
      }
      // On libère la mémoire mobilisée pour cette requête
      // $row reste accessible
      mysql_free_result ($db_result);

      // Deuxième script : on ajoute une ligne
      // On construit la date d'aujourd'hui
      // comme sql la construit
      $today = date("y-m-d");
      //On se connecte
      ConnexionBdd();
      //On prépare la commande sql
      $sql = "INSERT INTO TABLE_ETUDIANT
              VALUES (\"\", \"Nom1\", \"M\", \"".$today ."\", \"SRC\", 1124400, 0)";
      $sql2 = "DELETE FROM TABLE_ETUDIANT WHERE Telephone=1124400";
      // Envoie de la requête et arrêt du script s'il y a une erreur
      mysql_query ($sql) or mysql_query ($sql2) or die ("Erreur SQL !".$sql."<br />".mysql_error());
      // on ferme la connexion
      mysql_close();
      echo "<br>";

      // Troisième script script : on relit les informations
```

```
            //On se connecte
            ConnexionBdd();
            // On prépare la requête
            $sql = "SELECT * FROM TABLE_ETUDIANT WHERE Section=\"SRC\"";
            // Envoie de la requête et arrêt du script s'il y a une erreur
            $db_result = mysql_query($sql) or die ("Erreur SQL !".$sql."<br />".mysql_error());
            // Lecture du résultat ligne par ligne
            while ($data = mysql_fetch_array($db_result))
            {
              print_r($data);
              echo "<br>";
            }
            //On libère la mémoire mobilisée pour cette requête
            //$data reste accessibla
            mysql_free_result ($db_result);
            //On ferme sql
            mysql_close ();
          ?>
        </body>
      </html>
```

Utilisation d'un environnement objet

En plus de l'API MySQL, PHP fournit aussi une API générique, ou couche d'abstraction, pour gérer le SQL : **PHP Data Object (PDO)**. Le principal avantage de l'utilisation d'une couche d'abstraction est sa meilleure flexibilité : lors d'un changement de SGDB, il suffit de changer un paramètre lors de l'instanciation de l'objet PDO, le reste du code étant inchangé, au contraire d'une API spécifique.

PDO étant une API générique, il n'est pas étonnant d'apprendre qu'elle soit orientée objets. Pour obtenir plus d'informations, se référencer au site officiel `http://fr.php.net/manual/fr/book.pdo.php`. Nous retiendrons juste les trois classes qui la composent :

- La classe `PDO` : Représente une connexion entre PHP et un serveur de base de données.
- La classe `PDOStatement` : Représente une requête préparée et, une fois exécutée, le jeu de résultats associé.
- La classe `PDOException` : Représente une erreur émise par PDO. Une exception `PDOException` ne doit pas être lancée depuis notre code.

Une utilisation basique de de cette API peut se faire facilement grâce aux méthodes suivante :

- `PDO::__construct() ( string $dsn [, string $username [, string $password [, array $driver_options ]]] )` : Créer un objet PDO qui représente une **connexion à la base**. le `$dsn` contient les informations requises pour se connecter à la base (e.g. `"mysql:dbname=essai ; host=127.0.0.1"`).
- `PDO::prepare ( string $statement [, array $driver_options = array() ] )` : **Prépare une requête** pour l'exécution. Renvoie un objet `PDOStatement`.
- `PDOStatement::execute ([ array $input_parameters ] )` : **Exécute une requête** préparée. Renvoie un booléen pour prévenir de la bonne exécution.
- `PDOStatement::fetch ([ int $fetch_style = PDO::FETCH_BOTH [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )` : **Récupère une ligne** depuis un jeu de résultats associé à l'objet `PDOStatement`. Le paramètre `fetch_style` détermine la façon dont PDO retourne la ligne.
- `PDOStatement::fetchAll ([ int $fetch_style = PDO::FETCH_BOTH [, mixed $fetch_argument = [, array $ctor_args = array() ]]] )` : Retourne un tableau contenant toutes les lignes du jeu d'enregistrements.

Voici un exemple d'application de PDO, effectuant la même chose que le script précédent :

```
<?php
  // Définition préalable de fonctions utiles... et réutilisables...

  // Affiche joliment les exceptions PDO
  // Source : php.net
  function Afficher_Exception_PDO($err)
  {
    $trace = "<table border=\"0\">";
    foreach ($err->getTrace() as $a => $b) {
      foreach ($b as $c => $d) {
        if ($c == "args") {
          foreach ($d as $e => $f) {
            $trace .= "<tr><td><b>" . strval($a) . "#</b></td><td align=\"right\">
              <u>args:</u></td> <td><u>" . $e . "</u>:</td><td><i>" . $f . "</i></td></tr>";
          }
        } else {
          $trace .= "<tr><td><b>" . strval($a) . "#</b></td><td align=\"right\">
            <u>" . $c . "</u>:</td><td></td><td><i>" . $d . "</i></td>";
        }
      }
    }
    $trace .= "</table>";
    echo "<br /><br /><br />
      <font face= \"Verdana\">
      <center><fieldset style=\"width: 66%; border: 4px solid white; background: black;\">
      <legend><b>[</b>PHP PDO Error " . strval($err->getCode()) . "<b>]</b></legend>
      <table border=\"0\">
      <tr><td align=\"right\"><b><u>Message:</u></b></td><td><i>".$err->getMessage()."</i></td></tr>
      <tr><td align=\"right\"><b><u>Code:</u></b></td><td><i>".strval($err->getCode())."</i></td></tr>
      <tr><td align=\"right\"><b><u>File:</u></b></td><td><i>".$err->getFile()."</i></td></tr>
      <tr><td align=\"right\"><b><u>Line:</u></b></td><td><i>".strval($err->getLine())."</i></td></tr>
      <tr><td align=\"right\"><b><u>Trace:</u></b></td><td><br /><br />" . $trace . "</td></tr>
      </table>
      </fieldset></center>
      </font>";
  }

  // Affiche le dernier message d'erreur d'un PDOStatement
  function dernier_erreur($statement)
  {
    $tab = $statement->errorInfo() ;
    return $tab[2] ;
  }
?>

<html>
  <head>
    <title>Test connexion avec PDO</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>

  <body>

    <?php
      // Initialisation des variables pour l'insertion des données
      $today = date("y-m-d");
      $numero = 1124400 ;

      // Essai d'un code
      try
      {
        // Connexion à la base
        $db = new PDO("mysql:host=127.0.0.1;dbname=essai", "ROBERT", "lemotdepassederobert");
        // Configure les erreurs comme émettant des exceptions
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // Préparation des requêtes
        $select_etudiant = $db->prepare("SELECT * FROM TABLE_ETUDIANT WHERE Section=\"SRC\"");
        // Notons l'utilisaton de variable grâce à ? ou :nom_variable
```

```
            $insert_etudiant =
              $db->prepare("INSERT INTO TABLE_ETUDIANT VALUES (\"\", \"Nom1\", \"M\", ?, \"SRC\", ?, 0)");
            $delete_etudiant = $db->prepare("DELETE FROM TABLE_ETUDIANT WHERE Telephone= :numerotel");

            // Exécution du premier script
            $select_etudiant->execute();
            $etudiant = $select_etudiant->fetchAll() ;

            // Affichage
            print_r($etudiant);
            echo "<br><br>";
          }
          catch (PDOException $err)
          {
            Afficher_Exception_PDO($err) ;
          }

          // Configure les erreurs comme ne faisant rien sur l'exécution
          $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);

          // Insertion ou suppression de la ligne
          $insert_etudiant->execute(array($today, $numero))
            or ($delete_etudiant->execute(array(":numerotel" => $numero)) && $delete_etudiant->rowCount()>0)
            or die("Probleme survenue. <br/>
                     Dernière erreur sur l'insertion : " . dernier_erreur($insert_etudiant)."<br/>
                     Dernière erreur sur la suppression : " . dernier_erreur($delete_etudiant) );

          // Essai d'un code
          try
          {
            // Configure les erreurs comme émettant des exceptions
            $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

            // Exécution du premier script de selection et affichage
            $select_etudiant->execute() ;
            while ($etudiant2 = $select_etudiant->fetch())
            {
              print_r($etudiant2);
              echo "<br>";
            }

            $db = null; // "Déconnexion"
          }
          catch (PDOException $err)
          {
            Afficher_Exception_PDO($err) ;
          }

        ?>
      </body>
    </html>
```

Note :

- La POO autorisant le principe d'héritage, nous pourrions construire une classe fille qui nous permettrait de réduire le nombre de méthode à appeler à une seule (c.f. section 3.9). Par exemple, au lieu d'appeler `execute()` puis `fetchAll()`, nous pourrions construire une classe héritant de ces méthodes et en créer une nouvelle (e.g. `AffichageSelection()`) qui les gèrerait automatiquement. De plus, la notion de constructeur nous permettrait de créer des requêtes basiques au moment de la création d'un objet.
- Dans l'exemple précédent, nous avons vu la puissance de la préparation de requêtes sous PDO qui nous permettent de créer des **requêtes incluant des variables**.

5 Le développement en Patron de conception

5.1 Le Design Pattern MVC

Les applications que nous avons pu donner en exemple, jusqu'à maintenant, ne sont pas des meilleurs en terme de maintenance (et de sécurité). En effet, si nous sommes amenés à changer un nom de table, dans notre base de données, il faudra refaire les scripts de la section 4.5 en modifiant chaque appel à celle-ci. Chaque changement de ligne augmente la probabilité d'erreurs.

Pour remédier à ce problème, nous pouvons utiliser les fondements de la programmation orientée objets, notamment l'encapsulation et le polymorphisme. En effet, ces deux propriétés aident à avoir des classes hermétiques à la modification mais capables de s'adapter, au besoin.

Une utilisation méthodique de ces propriétés peut être effectuée grâce aux **Design Pattern** (patron de conception). Il en existe quelques un, mais nous nous intéresserons qu'à un seul : le patron de conception **Modèle-Vue-Contrôleur (MVC)**. Celui-ci permet de couper notre site en trois composantes différentes ayant chacune une utilité définie. Le travail de groupe en est grandement aidé puisque chaque composante, bien que reliée aux autres, est totalement indépendante. Les deux premiers groupes, composées d'informaticiens, peuvent s'occuper des composantes Contrôleur et Modèle. Et le troisième groupe, composé de Web Designer et d'intégrateur Web, peut travailler sur la Vue. Voici ce que décrivent ces composantes :

Le Modèle définit le comportement de l'application et le traitement des données. Les données ne sont associées à aucune présentation. Le modèle assure la cohérence des données et offre des méthodes de manipulation de ces dernières. ***C'est un ensemble de classes donnant accès aux données.***

La Vue est l'interface utilisateur. Elle présente le modèle, reçoit les événements et en avertit le contrôleur. Elle ne modifie pas le modèle et n'effectue aucun traitement. ***C'est la partie de l'application qui se charge de l'affichage des données.***

Le Contrôleur gère les événements en utilisant le modèle et invoquant les vues. ***Ces classes s'occupent de charger les modèles adéquats, d'appliquer les traitements nécessaires et d'envoyer, aux vues, les données demandées.***

Le design pattern Modèle-Vue-Contrôleur est donc un pattern architectural qui sépare les données (le Modèle), l'interface homme-machine (la Vue) et la logique de contrôle (le Contrôleur). ***L'objectif est d'utiliser des contrôleurs pour traiter les données, des modèles pour les récupérer et des vues pour les afficher.***

Un contrôleur principal, appelé **front controller** (contrôleur frontal), est utilisé pour centraliser l'ensemble des requêtes du client. Ainsi, un seul script PHP est exposé en HTTP, ce qui limite les problèmes de duplication de code et améliore la sécurité du site. C'est le front controller qui **aiguille le travail** vers les autres contrôleurs. Il permet de générer des événements lors d'une modification du modèle et d'indiquer à la vue qu'il y a nécessité d'une mise à jour. Un schéma de ce pattern est donné en figure 3.

Il faut tout de même faire attention : ce modèle de conception complique l'architecture du site.

5.1.1 Le Modèle

Dans notre cas, le modèle sera développé en SQL et PHP. Il définit l'**interaction avec la base de données** et les traitement associés. Il peut y avoir autant de modèle qu'il y a de table. L'utilisation

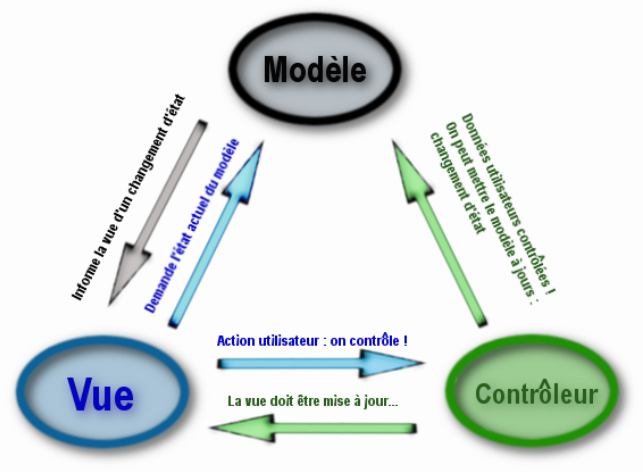

FIGURE 3 – Schéma du design pattern MVC

de l'encapsulation, proposée par le modèle objet du PHP, est fortement conseillée pour faciliter la maintenance.

Ce sont des objets qui font des traitements transparents pour l'utilisateur. **C'est le coeur du site.** Il est donc préférable de faire passer toutes les actions de l'utilisateur dans le contrôleur avant de les transmettre au modèle : il faut protéger le coeur. De plus, il faut évidement réfléchir à ce que notre modèle doit savoir faire (et ne pas faire) pour plus de sécurité.

5.1.2 La vue

La vue représente l'interface utilisateur, ce avec quoi il interagit. Elle n'**effectue aucun traitement et se contente d'afficher les données** que lui fournit le modèle, ou de transmettre des ordres venant de l'utilisateur. Il peut y avoir plusieurs vues présentant les données d'un même modèle. Dans le cas d'un site Web, la Vue peut être implantée en PHP, HTML, JavaScript, Flash...

Le développement des vues peut être facilité en se basant sur un fondement de la programmation objets : le polymorphisme.

5.1.3 Le Contrôleur

Le contrôleur gère l'interface entre le modèle, les vues et l'utilisateur : il interprète la requête de ce dernier et lui renvoie la vue correspondante. En d'autres termes, il effectue la **synchronisation entre le modèle et les vues** et permet le contrôle des données, avant de les envoyer au modèle. Pour l'implanter, nous utiliserons du PHP.

Dans les architectures Web PHP, nous retrouvons deux types de contrôleurs :

Le contrôleur principal ou front controller, c'est la seule entrée du site Web. Toutes les informations transitent par cette page, généralement appelée `index.php`, qui permet de choisir le contrôleur de page, l'action et les paramètres associées. C'est une étape importante de la sécurisation du site.

Le contrôleur de page gère une sous-commande du contrôleur frontal et permet la gestion d'une partie du modèle.

5.1.4 Exemple

Avant de commencer un exemple d'application de MVC, il est important de rappeler que l'implantation de ce design pattern, dans une application, nécessite une bonne conception dès le départ. Il faut bien réfléchir avant d'agir, ce qui peut prendre du temps. Cette méthode est donc déconseillée pour de petites applications. Cependant, dans un soucis de brièveté, nous ne donnerons qu'un petit exemple n'ayant pas nécessairement besoin d'un développement en MVC.

Nous utiliserons la base de données construite en section 2.2. Le site construit devra pouvoir gérer les connexions des utilisateurs et, lorsque cette action est effectuée, afficher une liste d'étudiants et d'enseignants. Cette liste devra être composée, si la personne connectée est un enseignant, de tous les étudiants et enseignants ou, si la personne est un étudiant, uniquement des étudiants et enseignants de sa section. De plus, nous souhaitons que les enseignants puissent supprimer un étudiant (et pas un autre enseignant).

Dans un soucis de simplification de la compréhension du modèle MVC, nous n'utiliserons pas l'aspect objet (entre autres, pour gérer notre base de données, nous utiliserons l'API MySQL...).

Pour commencer, il nous faut créer notre contrôleur principale, nommé `index.php`. Il sera chargé d'aiguiller et de contrôler les données vers les contrôleurs de pages. Nous pouvons, par exemple, utiliser quatre contrôleurs de pages :

Enseignant qui permettra de lister et d'administrer les enseignants.

Etudiant qui permettra de lister et d'administrer les étudiants.

Visiteur qui permettra d'afficher un formulaire de connexion, si l'utilisateur n'est pas connecté.

Erreur qui permettra de savoir ce qu'il faut faire en cas d'erreurs.

Un peu plus en détails :

- Le contrôleur `Enseignant` devra inclure les méthodes :
  - `read` : Permet l'affichage des données de tous les étudiants et enseignants.
  - `delete` : Permet la suppression d'un étudiant.
  - `deconnexion` : Permet la déconnexion de l'utilisateur.

- Le contrôleur `Etudiant` devra inclure les méthodes :
  - `read` : Permet l'affichage des données de tous les étudiants et enseignants d'une section.
  - `deconnexion` : Permet la déconnexion de l'utilisateur.

- Le contrôleur `Visiteur` devra inclure les méthodes :
  - `read` : Permet l'affichage du formulaire de connexion.
  - `connexion` : Permet la connexion d'un utilisateur.

- Le contrôleur `Erreur` devra inclure la méthode :
  - `read` : Permet l'affichage d'une page d'erreur.

Chaque méthode listée, permettant une action autre que l'affichage, doivent être implantées dans un modèle correspondant au contrôleur. De même, toutes fonctions devant afficher les données d'une `classe` doivent être définies dans une vue correspondante au contrôleur de pages. Pour mettre un peu d'ordre, nous pouvons utiliser l'arborescence suivante :

/index.php
/Config/config.php
/Config/gestion_bdd.php
/Config/style.css
/Controller/Enseignant.php
/Controller/Etudiant.php
/Controller/Visiteur.php
/Controller/Erreur.php
/Model/Enseignant.php
/Model/Etudiant.php
/Model/Visiteur.php
/Model/Erreur.php
/View/footer.php
/View/header.php
/View/Enseignant/display.php
/View/Etudiant/display.php
/View/Visiteur/display.php
/View/Erreur/display.php

Voici les sources :

/index.php

```
<?php
  /*
   * Ceci est le contrôleur principal, le seul point d'entrée vers notre site web
   */

  //definir la constante APPPATH vers le répertoire application
  define("APPPATH", realpath(dirname(__FILE__)).DIRECTORY_SEPARATOR);
  //inclure en utilisant APPPATH le fichier constants.php dans config/
  require_once APPPATH."Config".DIRECTORY_SEPARATOR."config.php";

  // On recherche le contrôleur adapté et on l'exécute
  if (isset($_GET["module"]))
    $module = $_GET["module"];
  else
    if (isset($_POST["module"]))
      $module = $_POST["module"];
    else
      if (isset($_COOKIE["nom"]))
        $module = $_COOKIE["statut"];
      else
        $module="Visiteur";

  require_once CONTROLLERSPATH.$module.".php";
?>
```

/Config/config.php

```
<?php
  /*
   * Dans un site web dynamique il faut toujours définir l'url pour charger les ressources
   * (script JS, CSS, images, ...) et
   * les identifiants de connexion vers la base de données.
   */

  // URL l'url du site à modifier en fonction du déploiement
  define("URL","http://127.0.0.1:8080/exemple_MVC/");

  // Les informations de connexion
```

```
  // DB_HOST l'adresse IP ou le nom de la machine qui héberge le SGBD
  define("DB_HOST","127.0.0.1");
  // DB_NAME le nom de la base de données
  define("DB_NAME","essai");
  // Le login sur la base
  define("DB_LOGIN","ROBERT");
  // Le mot de passe sur la base
  define("DB_PASSWORD","lemotdepassederobert");

  //CSSURL ou se trouve la css
  define("CSSURL",URL."Config/style.css");
  //CONTROLLERPATH où se trouvent les contrôleurs de page
  define("CONTROLLERSPATH",APPPATH."Controller".DIRECTORY_SEPARATOR);
  //VIEWSPATH où se trouvent les vues
  define("VIEWSPATH", APPPATH."View".DIRECTORY_SEPARATOR);
  define("VIEWSPATHENSEIGNANT", VIEWSPATH."Enseignant".DIRECTORY_SEPARATOR);
  define("VIEWSPATHERREUR", VIEWSPATH."Erreur".DIRECTORY_SEPARATOR);
  define("VIEWSPATHETUDIANT", VIEWSPATH."Etudiant".DIRECTORY_SEPARATOR);
  define("VIEWSPATHVISITEUR", VIEWSPATH."Visiteur".DIRECTORY_SEPARATOR);
  //MODELSPATH où se trouvent les modèles
  define("MODELSPATH",APPPATH."Model".DIRECTORY_SEPARATOR);
?>
```

/Config/gestion_bdd.php

```
<?php
  function ConnexionBdd()
  // Fonction de connexion et sélection de la base de données
  {
    $base = mysql_connect (DB_HOST, DB_LOGIN, DB_PASSWORD);
    mysql_select_db (DB_NAME, $base) ;
  }

  function DeconnexionBdd()
  // Ferme la bdd courante
  {
    mysql_close();
  }
?>
```

/Controller/Enseignant.php

```
<?php
  /*
   * Le contrôleur de page qui gère les Enseignants
   */

  // Gestion de la demande d'action
  if (isset($_GET["action"]))
    $action = $_GET["action"];
  else
    if (isset($_POST["action"]))
      $action = $_POST["action"];
    else
      $action="read";

  switch ($action)
  {
    case ("read"):
      require_once MODELSPATH."Enseignant.php";
      $etudiant = Enseignant\GetEtudiant();
      $enseignant = Enseignant\GetEnseignant();
      require_once VIEWSPATH."header.php";
      require_once VIEWSPATHENSEIGNANT."display.php";
      require_once VIEWSPATH."footer.php";
      break;
    case ("delete"):
      $TelEtudiant = $_POST["TelEtudiant"];
```

```
        require_once MODELSPATH."Enseignant.php";
        $personnes = Enseignant\delete($TelEtudiant);
        header("Location: ".URL."index.php?action=read");
        break;
      case ("deconnexion"):
        require_once MODELSPATH."Enseignant.php";
        $deco = Enseignant\deconnexion();
        if ($deco == false)
          header("Location: ".URL."index.php?module=Erreur&erreur=DECO");
        // Redirection de l'utilisateur
        header("Location: ".URL."index.php");
        break;
      default:
        header("Location: ".URL."index.php?module=Erreur&erreur=PHP");
        break;
    }
  ?>
```

/Controller/Etudiant.php

```
  <?php
    /*
     * Le contrôleur de page qui gère les Etudiants
     */

    // Gestion de la demande d'action
    if (isset($_GET["action"]))
      $action = $_GET["action"];
    else
      if (isset($_POST["action"]))
        $action = $_POST["action"];
      else
        $action="read";

    switch ($action)
    {
      case ("read"):
        if (isset($_COOKIE["section"]))
          $section_en_cours = $_COOKIE["section"] ;
        else
          if (isset($_GET["action"]))
            $section_en_cours = $_GET["action"];
          else
            header("Location: ".URL."index.php?module=Erreur&erreur=PHP");

        require_once MODELSPATH."Etudiant.php";
        $etudiant = Etudiant\GetEtudiant($section_en_cours);
        $enseignant = Etudiant\GetEnseignant($section_en_cours);
        if ($etudiant==false || $enseignant==false)
          header("Location: ".URL."index.php?module=Erreur&erreur=SQL");
        require_once VIEWSPATH."header.php";
        require_once VIEWSPATHETUDIANT."display.php";
        require_once VIEWSPATH."footer.php";
        break;
      case ("deconnexion"):
        require_once MODELSPATH."Etudiant.php";
        $deco = Etudiant\deconnexion();
        if ($deco == false)
          header("Location: ".URL."ndex.php?module=Erreur&erreur=SQL");
        // Redirection de l'utilisateur
        header("Location: ".URL."index.php");
        break;
      default:
        header("Location: ".URL."index.php?module=Erreur&erreur=PHP");
        break;
    }
  ?>
```

/Controller/Visiteur.php

```
<?php
  /*
   * Le contrôleur de page qui gère les Visiteurs
   */

  // Gestion de la demande d'action
  if (isset($_GET["action"]))
    $action = $_GET["action"];
  else
    if (isset($_POST["action"]))
      $action = $_POST["action"];
    else
      $action="read";

  switch ($action)
  {
    case "read":
      require_once VIEWSPATH."header.php";
      require_once VIEWSPATHVISITEUR."display.php";
      require_once VIEWSPATH."footer.php";
      break;
    case "connexion":
      $nom = $_POST["nom"];
      $type_utilisateur = $_POST["statut"] ;
      require_once MODELSPATH."Visiteur.php";
      $section = Visiteur\Connexion($nom,$type_utilisateur);
      if ($section!=false)
        header("Location: ".URL."index.php?module=".$type_utilisateur."&section=".$section );
      else
        header("Location: ".URL."index.php?module=Erreur&erreur=LOGIN");
      break;
    default:
      header("Location: ".URL."index.php?module=Erreur&erreur=PHP");
      break;
  }
?>
```

/Controller/Erreur.php

```
<?php

  $action="read";

  if (isset($_COOKIE["erreur"]))
    $erreur = $_COOKIE["erreur"] ;
  else
    if (isset($_GET["erreur"]))
      $erreur = $_GET["erreur"];
    else
      $erreur = "Erreur";

  switch ($action)
  {
    case "read":
      require_once MODELSPATH."Erreur.php";
      $messageerreur = Erreur\GererErreur($erreur);
      require_once VIEWSPATH."header.php";
      require_once VIEWSPATHERREUR."display.php";
      require_once VIEWSPATH."footer.php";
      break;
    default:
      header("Location: ".URL."index.php?module=Erreur&erreur=PHP");
      break;
  }
?>
```

/Model/Enseignant.php

```
<?php
  // L'espace de nommage est Enseignant
  // pour utiliser les fonctions il faut utiliser \Enseignant\nom_fonction
  namespace Enseignant;

  // Inclusion des utilitaires de connexion
  require_once APPPATH."Config".DIRECTORY_SEPARATOR."gestion_bdd.php";

  function GetEtudiant()
  {
    ConnexionBdd();
    $sql = "SELECT Nom, Telephone, Section FROM TABLE_ETUDIANT";
    $req = mysql_query($sql);
    if ($req==false)
      $res = false ;
    else
    {
      $res= array();
      while ($data = mysql_fetch_assoc($req))
      {
        $res[$data["Nom"]] = array("Nom"=>$data["Nom"], "Telephone"=>$data["Telephone"],
                                   "Section"=>$data["Section"]);
      }
    }
    DeconnexionBdd();
    return $res;
  }

  function GetEnseignant()
  {
    ConnexionBdd();
    $sql = "SELECT Nom, Mail, Section FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
            WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION.idEnseignant
            AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection";
    $req = mysql_query($sql);
    if ($req==false)
      $res = false ;
    else
    {
      $res= array();
      while ($data = mysql_fetch_assoc($req))
      {
        $res[$data["Nom"]] = array("Nom"=>$data["Nom"], "Mail"=>$data["Mail"],
                                   "Section"=>$data["Section"]);
      }
    }
    DeconnexionBdd();
    return $res;
  }

  function delete($tel)
  {
    $rst=false;
    ConnexionBdd();
    // Préparation de la requète
    $sql="DELETE FROM TABLE_ETUDIANT WHERE Telephone=\"".$tel."\"";
    $result=mysql_query($sql);
    DeconnexionBdd();
    return $result;
  }

  function deconnexion()
  {
    $rst = true ;
    if(empty($_COOKIE["nom"]))
      $rst = false ;
    else
    {
      // Suppression des cookies
```

```
            setcookie("nom", "", time() - 1, "/");
            setcookie("statut", "", time() - 1, "/");
            setcookie("section", "", time() - 1, "/");
          }
          return $rst ;
        }
      ?>
```

/Model/Etudiant.php

```
      <?php
        // L'espace de nommage est Etudiant
        // pour utiliser les fonctions il faut utiliser \Etudiant\nom_fonction
        namespace Etudiant;

        // Inclusion des utilitaires de connexion
        require_once APPPATH."Config".DIRECTORY_SEPARATOR."gestion_bdd.php";

        function GetEtudiant($section_en_cours)
        {
          ConnexionBdd();
          $sql = "SELECT Nom, Telephone, Section FROM TABLE_ETUDIANT where Section=\"".$section_en_cours."\"";
          $req = mysql_query($sql) ;
          if ($req==false)
            $res = false ;
          else
          {
            $res= array();
            while ($data = mysql_fetch_assoc($req))
            {
              $res[$data["Nom"]] = array("Nom"=>$data["Nom"], "Telephone"=>$data["Telephone"],
                                         "Section"=>$data["Section"]);
            }
          }
          DeconnexionBdd();
          return $res;
        }

        function GetEnseignant($section_en_cours)
        {
          ConnexionBdd();
          $sql = "SELECT Nom, Mail, Section FROM TABLE_ENSEIGNANT, TABLE_ASSOCIATION, TABLE_SECTION
                  WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION.idEnseignant
                  AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
                  AND Section=\"".$section_en_cours."\"";
          $req = mysql_query($sql) ;
          if ($req==false)
            $res = false ;
          else
          {
            $res= array();
            while ($data = mysql_fetch_assoc($req))
            {
              $res[$data["Nom"]] = array("Nom"=>$data["Nom"], "Mail"=>$data["Mail"],
                                         "Section"=>$data["Section"]);
            }
          }
          DeconnexionBdd();
          return $res;
        }

        function deconnexion()
        {
          $rst = true ;
          if(empty($_COOKIE["nom"]))
            $rst = false ;
          else
          {
            // Suppression des cookies
            setcookie("nom", "", time() - 1, "/");
```

```
            setcookie("statut", "", time() - 1, "/");
            setcookie("section", "", time() - 1, "/");
          }
          return $rst ;
        }
      ?>
```

/Model/Visiteur.php

```
      <?php
        // L'espace de nommage est Visiteur
        // pour utiliser les fonctions il faut utiliser \Visiteur\nom_fonction
        namespace Visiteur;

        // Inclusion des utilitaires de connexion
        require_once APPPATH."Config".DIRECTORY_SEPARATOR."gestion_bdd.php";

        function Connexion($nom,$type_utilisateur)
        {
          $rst=false;
          ConnexionBdd();
          // Préparation de la requète
          switch ($type_utilisateur)
          {
            case "Etudiant":
              $sql="select Section from TABLE_".$type_utilisateur." where nom= \"".$nom."\"";
              break;
            case "Enseignant":
              $sql="SELECT Section FROM TABLE_".$type_utilisateur.", TABLE_ASSOCIATION, TABLE_SECTION
                    WHERE TABLE_ENSEIGNANT.idEnseignant=TABLE_ASSOCIATION.idEnseignant
                    AND TABLE_SECTION.idSection=TABLE_ASSOCIATION.idSection
                    AND nom= \"".$nom."\"";
              break;
            default:
              header("Location: ".URL."index.php?module=Erreur&erreur=PHP");
          }
          $result=mysql_query($sql) ;
          if ($result==false)
            $res = false ;
          else
          {
            if (mysql_num_rows ( $result )==0)
              $rst=false;
            else
            {
              $row = mysql_fetch_array($result, MYSQL_NUM);
              $section = $row[0] ;

              $expiration =  !(empty($_GET["connexion_auto"])) ? 0 : time() + 30 * 24 * 60 * 60; // 30j
              setcookie("nom", $nom, $expiration, "/");
              setcookie("statut", $type_utilisateur, $expiration, "/");
              setcookie("section", $section, $expiration, "/");

              $rst=$section;
            }
          }
          DeconnexionBdd();
          return $rst;
        }
      ?>
```

/Model/Erreur.php

```
      <?php
        // L'espace de nommage est Erreur
        // pour utiliser les fonctions il faut utiliser \Erreur\nom_fonction
        namespace Erreur;
```

```
  function GererErreur($erreur)
  {
    switch($erreur)
    {
      case "SQL":
        $res = "L'erreur proviens de la gestion de base de données" ;
        break;
      case "PHP":
        $res = "L'erreur proviens du code PHP" ;
        break;
      case "LOGIN":
        $res = "L'utilisateur n'existe pas" ;
        break;
      default:
        $res = "L'erreur est inconnue" ;
        break;
    }
    return $res ;
  }
?>
```

/View/footer.php

```
  </body>
</html>
```

/View/header.php

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <title> Exemple MVC </title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link href="<?php echo CSSURL;?>" rel="stylesheet" type="text/css" media="screen" />
  </head>
  <body>
    <?php echo "L'utilisateur actuel est un ".$module."<br>" ; ?>
```

/View/Enseignant/display.php

```
<p> Se deconnecter </p>
<form action="<?php echo URL."index.php?action=deconnexion" ?>" method="post" >
  <p>
    <input type="submit" name="deconnexion" value="Deconnexion">
  </p>
</form>
<p> Etudiant </p>
<?php
  $tab_ligne=array_keys($etudiant);
  $tab_colonne=array_keys($etudiant[$tab_ligne[0]]);

  $text="";
  $text.="<form action=\"".URL."index.php?action=delete\" method=\"post\" >";
  $text.="<table border=\"1\">
          <tr>";

  $i=0;
  while ($i<count($tab_colonne))
  {
    $text.= "<th>".$tab_colonne[$i]."</th>";
    $i+=1;
  }
  $text.= "<th> Delete </th>";
  $text.= "</tr>";
```

```
  $i=0;
  while ($i<count($tab_ligne))
  {
    $ligne_en_cours = $etudiant[$tab_ligne[$i]];
    $text.=  "<tr>";
    $ii = 0 ;
    while ($ii<count($tab_colonne))
    {
      $text.= "<th>".$ligne_en_cours[$tab_colonne[$ii]]."</th>";
      $ii+=1;
    }
    $text.="<th>
              <INPUT name=\"TelEtudiant\" type=\"submit\" value=\"".$ligne_en_cours["Telephone"]."\">
            </th>";
    $text.= "</tr>";
    $i++;
  }
  $text.= "</tr>";
  $text.="</tr>
          </table> <br>";
  $text.="</form>";
  echo $text."<br>";
?>

<p> Enseignant </p>
<?php
  print_r($enseignant);
  echo "<br>";
?>
```

/View/Etudiant/display.php

```
<p> Se deconnecter </p>
<form action="<?php echo URL."index.php?action=deconnexion" ?>" method="post" >
  <p>
    <input type="submit" name="deconnexion" value="Deconnexion">
  </p>
</form>
<p> Étudiant </p>
<?php
  print_r($etudiant);
  echo "<br>";
?>
<p> Enseignants </p>
<?php
  print_r($enseignant);
  echo "<br>";
?>
```

/View/Visiteur/display.php

```
<p> Se connecter </p>
<form action="<?php echo URL."index.php?action=connexion" ?>" method="post" >
  <p>
    Nom <input type="text" name="nom"/><br>
    Statut <SELECT name="statut" size=1 >
      <option value="Enseignant" >Enseignant
      <option value="Etudiant" selected>Étudiant
    </SELECT>
    <input type="submit" name="connexion" value="Connexion">
  </p>
</form>
```

/View/Erreur/display.php

```
<p> Une erreur est survenue </p>
```

```
<?php
  echo $messageerreur ;
?>

<p> Retour a l'accueil </p>
<form action="<?php echo URL."index.php" ?>" method="post" >
  <p>
    <input type="submit" name="retour" value="Accueil">
  </p>
</form>
```

Note :
- L'ajout d'un répertoire de configurations nous sert à stocker la déclaration des constates permettant de gérer notre arborescence. Il contient aussi des fonctions de gestion de la base de données.
- Ce programme d'exemple, bien que donnant un aperçu de l'utilisation du modèle MVC donne aussi un certains nombre de choses à ne pas faire. En effet, à plusieurs reprise, nous utilisons des données provenant de l'utilisateur (`GET`, `POST`, `COOKIES`), sans les filtrer, ce qui constitue une faille de sécurité. De même, l'utilisateur ne devrait pas avoir accès au dossier de l'architecture...
- Un autre problème de sécurité est aussi dû au non chiffrement de l'URL du site.
- L'utilisation d'un contrôleur `Personne`, en plus de l'utilisation de deux contrôleurs `Etudiant` et `Enseignant` aurait peut être été plus judicieux. En effet, dans notre cas, nous n'utilisons pas le modèle objet du PHP, ce qui rends nos développement plus lourd (copie de code).
- La programmation objet aurait aussi pu être utilisée pour gérer notre base de données. Il est préférable d'utiliser le PDO à la place de l'API MySQL pour faciliter la maintenance.

5.2 L'utilisation de Framework

Nous avons pu voir, dans les exemples précédent, l'utilisation de fonctions permettant d'inclure des fichiers dans nos script, comme `require`, `include`, `require_once`, `include_once`.

`include` contrairement à `require` ne conduit pas à une erreur en cas de non chargement du script. De plus, à force d'inclure des scripts contenus dans d'autres, une boucle peut se produire et des doubles définitions apparaître, causant des erreurs. `require_once` et `include_once`, ne conduisent pas à une erreur en cas de double définition.

Cependant, inclure des sous-scripts peut très vite se révéler contraignant et pas toujours utile. Heureusement, il existe des outils pour faciliter la programmation. En effet, il n'y a pas besoin de réécrire des fonctions courantes et, pour éviter de le faire, il existe des **Framework** : c'est un ensemble d'outils et de composants logiciels qui suivent un plan d'architecture rigoureux et sont développés selon des modèles de conception.

Un premier Framework pouvant être utilisé est CakePHP. Il est développé en PHP et est basé sur le modèle de conception MVC. Plus d'informations peuvent être trouvées sur le site `www.cakephp.org`.

CodeIgniter est un Framework léger et très complet et basé, lui aussi, sur le modèle de conception MVC. Il permet, par exemple, le cryptage des données. Des informations peuvent être trouvées sur le Web : `http://codeigniter.com/`.

Chaque Framework à ces qualités et ces défauts. Certains permettent une gestion très intéressantes des bases de données puisque basées sur un ORM. C'est le cas du Framework Symfony (`http://www.symfony-project.org/`), qui est un Framework MVC, libre, écrit en PHP 5, contenant un ORM par defaut, l'ORM doctrine (`http://www.doctrine-project.org/`).

Un **ORM**, pour object-relational mapping (ou mapping objet-relationnel), permet d'utiliser une base de données relationnelle comme si elle était orientée objets.

Bien d'autres Framework existent pour développer une application, et pas seulement pour créer un site en PHP. Nous pouvons, par exemple, citer le Framework Script.aculo.us (disponnible à l'adresse `Script.aculo.us`) qui permet d'utiliser du Javascript facilement, et notamment de l'AJAX qui est une manière de développer des pages Web en se basant sur les technologies suivantes :
- HTML et CSS pour la présentation.
- DOM (Document Object Model) pour la représentation en objets de la page Web.
- Javascript et en particulier l'objet `XMLHTTPRequest` pour manipuler des requêtes et des réponses.

6 La sécurité

6.1 Les attaques sur le SGDB

Les éléments de la base de données, venant d'une source externe au script, doivent être traités de la même manière que les entrées d'un formulaire : avec prudence. Il faut les valider et les filtrer. Nous allons montrer quelques exemples de ce qui peut arriver si nous ne le faisons pas.

6.1.1 L'injection SQL

Une injection SQL est un type d'exploitation d'une faille de sécurité d'une application interagissant avec une base de données. Elle est possible en injectant une requête SQL non prévue par le système et pouvant compromettre sa sécurité.

Dans le cas d'un site Web dynamique, à base de PHP, ne vérifiant pas les données entrantes, nous pourrions, par exemple, se connecter sans mot de passe.

Prenons le cas d'un formulaire de connexion demandant le pseudonyme et le mot de passe. Dans un cas normal, une fois le formulaire complété, PHP envoie la requêtes SQL suivante pour vérifier les données :

```
SELECT Id FROM Utilisateur WHERE name = "LeNomSaisi" AND password = "LeMotDePasse";
```

Si nous souhaiterions nous connecter sans mot de passe (nous l'avons oublié), nous pourrions entrer un nom de la forme : `Pseudo" --`. Alors, nous pourrions nous connecter avec n'importe quel mot de passe. En effet, la requête deviendrait :

```
SELECT Id FROM Utilisateur WHERE name = "Pseudo" --" AND password = "LeMotDePasse";
```

En lisant attentivement, nous nous rendons compte que le moteur SQL exécutera la requête :

```
SELECT Id FROM Utilisateur WHERE name = "Pseudo"
```

En effet, le signe `--` signifie le début d'un commentaire et toutes les informations suivant ce caractère ne seront pas lues. La requête étant non vide, si le nom `Pseudo` existe, nous nous connecterons sans mot de passe.

Si cette première méthode ne fonctionne pas, et que nous souhaitons réellement récupérer nos données (correspondantes à notre pseudo), sans mot de passe (que nous avons oublié), nous pouvons aussi entrer notre identifiant avec un mot de passe de la forme `" or 1=1 --`. La requête alors exécutée sera :

```
SELECT Id FROM Utilisateur WHERE name = "Pseudo" AND password = "" OR 1=1 --";
```

La condition `1=1` étant toujours vrai, nous n'avons donc pas à nous soucier du mot de passe.

Pouvoir récupérer nos données sans mot de passe peut être pratique, mais ce n'est pas toujours le point de vu du développeur, qui peut utiliser des méthodes afin d'empêcher cela, comme :

- **Traiter correctement les chaînes de caractères** entrées par l'utilisateur. En PHP nous pouvons utiliser la fonction `mysql_real_escape_string`, qui transformera la chaîne `" --` en `\" --`. Ce qui aura pour conséquence d'annuler l'effet de l'apostrophe et la marque de commentaire sera alors inclue dans la chaîne du pseudo.
- Utiliser la fonction `ctype_digit` pour **vérifier les variables numériques** des requêtes. Nous pouvons aussi forcer la transformation de la variable en nombre en la faisant précéder d'un transtypeur, comme `(int)` si on attend un entier.

L'utilisation de `ctype_digit` doit se faire de paire avec `mysql_real_escape_string` puisque nous pouvons faire une injection SQL à partir de variable numérique ou de chaînes de caractères. Par exemple, pour effectuer une injection SQL sur une variable numérique, et en supposant que le mot de passe doit être un entier, nous pourrions entrer `0 or 1=1` et la requête SQL sera toujours bonne puisque l'on aura :

```
SELECT Id FROM Utilisateur WHERE name = "Pseudo" AND password = 0 OR 1=1 ;
```

La fonction `ctype_digit` nous sera alors utile puisqu'elle permet de forcer le nombre entré à être un entier. Ainsi `0 or 1=1` deviendra `0`.

En règle générale, pour éviter une injection SQL, il faut :

- **Utiliser des requêtes SQL paramétrées** (requêtes à trous envoyées au serveur SQL, serveur à qui nous envoyons, par la suite, les paramètres qui boucheront les trous), ainsi c'est le SGBD qui se charge d'échapper les caractères selon le type des paramètres. La classe PDO permet de le faire facilement.
- **Vérifier, de manière précise et exhaustive, l'ensemble des données venant de l'utilisateur**. Nous pouvons, par exemple, utiliser une **expression rationnelle** afin de valider qu'une donnée entrée par l'utilisateur est bien de la forme souhaitée (c.f. `http://www.commentcamarche.net/contents/php/phpreg.php3`).
- **Utiliser des comptes utilisateurs SQL à accès limité** (en lecture seule sur certaines tables) quand cela est possible.
- Utiliser, en conjonction avec `mysql_real_escape_string`, la fonction `sprintf()` (pour typer les données).
- Utiliser, la fonction `stripslashes()` qui permet la suppression des antislashs d'une chaîne.

Note : En PHP, la classe PHP Data Objects permet d'abstraire la connexion à la base de données, et introduit un mécanisme de requêtes préparées résistant aux injections avec la méthode `prepare`.

6.1.2 Blind SQL Injection

Nous avons vu comment faire des injections SQL dans le cas de chaînes de caractères et de nombres, mais qu'en est-il des requêtes de type booléenne ?

Bien entendu, il existe une méthode pour ce type de requêtes : les **Blind SQL Injections** (injections SQL à l'aveugle). Ce type d'injection peut être utilisée dans le cadre d'un script renvoyant

vrai ou faux (e.g. un formulaire d'identification). Dans ce cas, nous savons juste si notre injection a fonctionnée, ou non, ce qui les rends difficile à pratiquer.

Généralement, il s'agit d'attaque brutale consistant a essayer d'obtenir un résultat tant que nous n'y sommes pas arrivés (e.g. recherche d'un mot de passe en utilisant toutes les combinaisons possibles). Il est évidement conseillé d'utiliser cette méthode avec intelligence. En effet, si nous recherchons notre mot de passe, égaré, et que nous savons que nous adorons notre chat, nous pourrons d'abord essayer avec son prénom, sa date de naissance... Nous pourrions gagner du temps. Et si nous détestons les chats, il y a plein de listes de mots de passes courants sur le net...

6.1.3 SQL Smuggling

Imaginons que nous avons oublié (encore une fois) notre mot de passe et que, malheureusement pour nous, le développeur à pris toutes les mesures pour empêcher une injection SQL. Au lieu de pleurer sur la perte de nos photos extrêmement importante de la dernière soirée étudiante, il existe encore une solution que nous pouvons exploiter pour les retrouver : le **SQL Smuggling**.

Cette méthode repose sur l'interprétation des caractères (parfois fausse) faite par certains SGDB, d'où le nom Smuggling signifiant contrebande. La mauvaise interprétation des caractères est d'autant plus dangereuse que l'attaque passe aisément le contrôle du pare-feu applicatif Web puisqu'il n'interprète pas les requêtes. Le but est alors de trouver des caractères non pris en charge par le SGDB (et qui seront donc convertis), sans pour autant être inconnus du pare-feu.

La visualisation de ce problème peut être faite grâce à des homoglyphes : ce sont des symboles qui ressemblent fortement à des lettres de l'alphabet occidental, mais qui ont une représentation Unicode [2] différente.
Dans le cas d'une base de données effectuant une conversion des caractères, le caractère U-02BC pourra passer outre le filtrage du pare-feu Web et être converti en son équivalent le plus proche : un guillemet simple (U-0027)...

Le développeur se souciant de cette traduction automatique peut :
- Ne pas autoriser la traduction automatique des caractères Unicode non supportés.
- Privilégier une approche par liste blanche des caractères supportés.
- Changer de SGDB pour choisir un système levant une erreur lorsqu'un caractère n'est pas reconnu.

6.2 Attaque du serveur Web

6.2.1 Le Cross Site Scripting (XSS)

Le cross-site scripting, abrégé **XSS**, est un type de faille de sécurité des sites Web. On peut le trouver dans les applications Web pouvant être utilisées par un attaquant pour provoquer un comportement du site Web différent de celui désiré par le créateur de la page (redirection vers un site, vol d'informations, etc.). Il est abrégé XSS pour ne pas être confondu avec le CSS (feuilles de style), X étant une abréviation commune pour "*cross*", croix en anglais.

Pour tester si un site est attaquable par XSS, il faut injecter des données arbitraires dans un site Web (e.g. déposer un message dans un forum) ou par des paramètres d'URL. Si ces données arrivent telles quelles dans la page Web transmise au navigateur, alors il existe une faille : nous pouvons nous

2. `http://www.utf8-chartable.de/`

en servir pour faire exécuter du code malveillant, en langage de script (du JavaScript le plus souvent), par le navigateur Web qui consulte cette page.

La détection de la présence d'une faille XSS peut se faire, par exemple, en entrant un script Javascript, dans un champ de formulaire (ou dans une URL), de la forme :

```
<script type="text/javascript">alert("bonjour")</script>
```

Si une boîte de dialogue apparaît, l'application Web est sensible aux attaques de type XSS.

L'exploitation d'une faille de type XSS permettrait à un intrus de réaliser les opérations suivantes :
- Redirection (parfois de manière transparente) de l'utilisateur.
- Vol d'informations, par exemple de sessions et cookies.
- Actions sur le site faillible, à l'insu de la victime et sous son identité (e.g. envoi de messages, suppression de données)
- Rendre la lecture d'une page difficile (e.g. boucle infinie d'alertes).

Pour éviter cette faille, plusieurs techniques existent :
- **Retraiter systématiquement le code HTML** produit par l'application avant l'envoi au navigateur.
- **Filtrer les variables** affichées ou enregistrées avec des caractères < et >. En règles générales, il faut éviter d'utiliser des données provenant de l'extérieur, au sein d'une chaîne qui sera exécutée, sans les avoir filtrées préalablement.
- En PHP :
  - Utiliser la fonction `htmlspecialchars()` qui, entre autres, filtre les < et >.
  - Utiliser la fonction `htmlentities()` qui est identique à `htmlspecialchars()` sauf qu'elle filtre tous les caractères équivalents au codage HTML ou Javascript.
  - Utiliser `strip_tags()` qui supprime les balises.
  - Utiliser `intval()` qui retourne la valeur numérique entière d'une variable.

6.2.2 Le Cross Site Tracing (XST)

Le Cross Site Tracing, abrégé **XST**, est une manière d'exploiter les failles de sécurité de type XSS, mettant à profit la méthode TRACE du protocole HTTP. La méthode TRACE [3] est utile lors de debogage, pour afficher les headers de la requêtes (e.g. cookies, données de formulaires).

De plus, une faille de type XSS ne permet pas, normalement, d'accéder aux informations relatives à un autre site que celui faillible (e.g. les cookies), à cause des modèles de sécurité des navigateurs. Le XST permet de contourner cette limitation en se servant de la méthode TRACE qui a pour fonction de renvoyer au client l'intégralité de l'en-tête de sa requête. Invoquée dynamiquement en JavaScript (grâce à `XMLHttpRequest` [4] par exemple), elle permet, dans certains cas, de récupérer les cookies relatifs au domaine ciblé par la requête. En effet, ceux-ci apparaissent automatiquement dans l'en-tête envoyée par le navigateur, et donc retournée par le serveur.

6.2.3 Les attaques de formulaire

Il faut rappeler qu'il y a plusieurs moyens d'envoyer les données d'un formulaire :

3. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html`
4. `http://www.w3.org/TR/XMLHttpRequest/` ; `http://www.commentcamarche.net/faq/18745-utilisation-de-l-objet-xmlhttprequest`

- `GET` : Les données transiteront par l'URL. Nous pouvons les récupérer grâce à l'array `$_GET`. Cette méthode est assez peu utilisée car nous ne pouvons pas envoyer beaucoup d'informations via l'URL.
- `POST` : Les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse. Cette méthode permet d'envoyer autant de données que nous souhaitons, ce qui fait qu'elle est, le plus souvent, privilégiée. Néanmoins, les données ne sont pas plus sécurisées qu'avec la méthode `GET`.

Il est assez simple de feinter une méthode `GET`. En effet, il suffit de lire la barre d'adresse du navigateur et de changer les données nous intéressant. Pour éviter ce problème, nous pouvons utiliser du cryptage de données, avant de les envoyées dans l'URL.

Le contournement d'une méthode `POST` est un peu plus difficile, mais tout aussi faisable. Il suffit de récupérer le code du formulaire (situé sur le serveur), grâce à notre navigateur Web. Avec un peu de reverse ingeniering, il faut comprendre comment il fonctionne et en recréer un (sur un autre serveur, ou sur notre machine local) permettant de transmettre des informations aux serveur d'origine (pour cela, il suffit de modifier le champ `action=` de la balise `form` pour faire pointer le formulaire vers l'url du site visé).

6.3 Faille dans la logique d'application

6.3.1 Le verrou mortel en SQL

Le verrou mortel est un phénomène qui se produit lorsque deux (ou plus) utilisateurs veulent accéder aux mêmes ressources. En effet, nous avons vu que les SGDB peuvent utiliser des transactions et poser des verrous pour éviter de mauvaises lectures, en fonction des options d'isolations demandées (c.f. section 2.5). Cependant, ces verrous peuvent poser problèmes, dans certains cas, et ne jamais être levés, causant un blocage du serveur : c'est **le verrou mortel**.

L'empêchement de la suppression des verrous peut être dû à une perte du client n'envoyant pas la commande de fin de transaction. Nous pouvons aussi voir ce problème lors d'un afflux trop important de requêtes. Dans ce cas là, le serveur les ajoutes dans une file d'attente et chaque utilisateur attend qu'un autre libère des ressources, ce qui peut entrainer un blocage. Un autre cas de verrou mortel peut arriver lorsque deux utilisateurs veulent accéder à des données croisées. Par exemple, deux utilisateurs veulent effectuer des opérations dans l'ordre suivant :

- Utilisateur 1
  1. SELECT sur la table 1
  2. UPDATE sur la table 2
  3. UPDATE sur la table 1
- Utilisateur 2
  1. SELECT sur la table 2
  2. UPDATE sur la table 1
  3. UPDATE sur la table 2

Ici, il y a risque de blocage puisque l'utilisateur 1 commence par verrouiller la table 1 et l'utilisateur 2, la table 2. Quand l'utilisateur 1 va vouloir utiliser la table 2, il ne pourra pas et le deuxième utilisateur sera dans le même cas avec la table 1. Ce qui provoque un blocage.

Des solutions existent pour éviter ce verrou mortel :

- La logique transactionnelle doit toujours être exécutée au plus près du serveur et non sur le poste client (e.g. il ne faut pas attendre que l'utilisateur clic sur un bouton).
- Le niveau d'isolation des transactions doit être choisi judicieusement.
- La manipulation des tables, dans les procédures stockées, comme dans le code client, doit toujours se faire dans le même ordre (e.g. l'ordre alphabétique du nom de table).
- Certains SGBD (comme MS SQL Server) sont assez sujet à l'interblocage et le seul remède est, en général, de "tuer" un utilisateur. Il existe des SGDB (comme Oracle ou InterBase) doté d'un algorithme qui empêche tout interblocage (e.g. un *time out*).

6.3.2 Les failles de PHP

PHP est de loin sans failles. La majeure partie est référencée sur le site `http://www.phpsecure.info`.

Voici un exemple de failles :
- La vulnérabilité "escape shell" (c.f. `http://www.ouah.org/escape-shell.txt`).
- L'utilisation d'une fonction `include()` avec des paramètres pouvant accéder à des fichiers "interdit" (comme le `.htacces` ou les fichiers log d'Apache).
- La fonction `mail()` permettant de spammer anonymement.
- Les modifications de script d'upload.

6.3.3 Les failles utilisant le couplage PHP-SGDB

Le serveur du site peut aussi être mis hors service en utilisant des actions comme l'appel à des requêtes MySQL multiples, ce qui constitue une **attaque DoS** (Denial of Service). En effet, dans une architecture classique, le serveur HTTP et le SGBD sont deux machines différentes, sur un même réseau local. Nous pouvons donc en déduire que, généralement, les deux applications (e.g. Apache et MySQL) communiquent via le protocole TCP/IP. Ainsi chaque requête transite par le réseau. Il est donc facile de le congestionner en envoyant, fréquemment, de lourdes requêtes. C'est pourquoi il est déconseillé d'utiliser une requête `SELECT *`, puisqu'elle retourne plus de données que nécessaire.

Pour aller plus loin

Beaucoup de choses sont encore à voir, notamment au niveau de la sécurité, nous pouvons citer :
- Le Response Splitting : `http://blog.4j4x.net/?p=15`.
- L'upload de fichier sécurisé : `http://www.vulgarisation-informatique.com/upload-php.php`.
- La sécurisation des données transmisent grâce au cryptage ou, par exemple, à SOAP : `http://www.soapuser.com/fr/basics1.html`.
- L'utilisation d'expressions régulières pour modifier et lire des chaînes de caractères : `http://www.commentcamarche.net/contents/php/phpreg.php3`.
- La protection du fichier `.htacces` : `http://www.siteduzero.com/tutoriel-3-14649-proteger-un-dossier-avec-un-htaccess.html`

De plus, d'autres outils peuvent être pratique pour la construction et la maintenance de site Web dynamique :
- L'utilisation du xml : `http://www.siteduzero.com/tutoriel-3-33440-le-point-sur-xml.html`
- L'utilisation de Flash, ou du HTML5, pour l'affichage.
- L'utilisation de l'URL rewriting pour faciliter l'indexation : `http://www.webrankinfo.com/dossiers/debutants/url-rewriting`.

Pour terminer, voici quelques liens utiles :
- `http://fr.php.net/manual/fr/`
- `http://www.lephpfacile.com/cours/4-les-variables-predefinies`
- `http://www.certa.ssi.gouv.fr/`

Table des figures

Liste des tableaux

A TD1 : Utilisation d'une base de données

Objectif :

Test de connaissance en SQL.

Exercice :

1. Proposer un modèle de conception de données qui modélise le problème suivant :

   Gestion d'un Agenda
   Plusieurs personnes pourront utiliser l'agenda, ils auront dans ce cas chacun leur propre planning. Les éléments contenus dans l'agenda pourront être des rendez-vous, des réunions ou des évènements. Chaque élément est soit privé (visible de l'utilisateur seul), soit public (visible de tous) ; il peut aussi nécessiter une présence de l'utilisateur ou non.
   Un rendez vous est caractérisé par une date de début, une date de fin, un objet, un résumé et une catégorie. Une réunion sera caractérisée, en plus, par la liste des participants (élément commun a plusieurs utilisateurs). Un évènement correspond à une date importante (par exemple, un anniversaire) et ne fait participer qu'un utilisateur.
2. Construire la base de données.
3. Donner un script permettant de sélectionner tous les rendez-vous privés d'un utilisateur, pour une période donnée (par exemple, le mois à venir).
4. Lister tous les participants d'une réunion donnée.
5. Compter le nombre de participants d'une réunion donnée.
6. Lister et afficher les utilisateurs, avec leurs réunions, même s'il n'en ont pas.
7. Modifier une réunion pour annuler la participation d'une personne.
8. Supprimer une réunion (penser à l'option de table **ON DELETE**).
9. Créer un **TRIGGER** permettant de modifier le mot de passe d'un utilisateur, lors de son ajout, si il est vide (penser aux pseudo table **NEW** et **OLD** dans un **TRIGGER** contenant les données en cours et avant modification).

Note : Pour commencer, vous pouvez faire ces requêtes sur les évènements.

B TD2 : Construction de page Web dynamique en PHP

Objectif :

Test de connaissance en PHP.

Exercice :

1. Réaliser une page, en langage PHP, qui affiche les tables de multiplications de 1 à 12, dans un tableau.
2. Réaliser une ou plusieurs pages, en langage PHP, qui permettent de faire quelques calculs simples d'une calculatrice.
3. Réaliser une ou plusieurs pages, en langage PHP, qui demandent à l'utilisateur son nom puis affiche la date du jour et un message contenant le nom entré par l'utilisateur.
4. Permettre à un utilisateur d'uploader et d'afficher le contenu d'un fichier.
5. Réaliser une fonction qui retourne le texte HTML affichant le calendrier d'un mois et d'une année donnée et utiliser la dans une page.
6. Créer et utiliser une classe PHP permettant de gérer un utilisateur d'agenda (nom, prénom, connection...).

C TD3 : Interaction avec SQL

Objectif :

Test d'utilisation du SQL intégré.

Exercice :

1. Utiliser la base de données du TD1 et l'API MySQL de PHP pour effectuer des opérations de bases sur l'agenda (ajouter des réunions, les visualiser...).
2. Utiliser PDO pour obtenir le même résultat.
3. Modifier les pages pour permettre l'affichage de l'agenda sous différentes formes (jour, semaine, mois...).
4. Modifier les pages (et la base de données) pour ajouter un administrateur pouvant ajouter/modifier un utilisateur.

D TD4 : Utilisation du design pattern Modèle-Vue-Contrôleur

Objectif :

Test d'utilisation du design pattern MVC.

Exercice :

1. Reprendre l'exemple de la section 5.1.4 en utilisant de la programmation objet.
2. Construire un site permettant la gestion des élèves absents.

Note : Vous pouvez vous inspirer de l'exemple de la section 5.1.4 et du TD3.

E TD5 : Sécurité d'un site

Objectif :

Essai de contournement d'un site web dynamique.

Exercice :

1. Aller sur le site `http://thibault-maillot.netne.net/TD5_securite/` et se connecter (login et mot de passe donnés en cours).
2. Essayer d'utiliser un utilisateur, déjà inscrit, ayant les droits d'administration.
3. Essayer de créer un utilisateur, avec les droits d'administration.
4. Embêter les autres pour les retarder.

Note :
- Penser à lire le cours...
- Utiliser la fonction de votre navigateur permettant de voir le code HTML de la page. Elle pourrait être utile pour deviner l'architecture du site.
- Pour la question 2, une injection SQL serait peut-être adaptée.
- Pour la question 3, un envoie de fausses données pourrait être adapté.
- Pour la question 4, bien qu'un utilisateur soit fortement embêté par du Flash, une solution plus simple pourrait être l'utilisation du JavaScript.