Emsi IIR°3
Année 2014-201

Le langage PHP5 et MySQL

Rabii. Elghorfi (rabii.elghorfi@outlook.com)

Plan

- Introduction à PHP
- Les Variables et les constantes
- Les Types
- Les Opérateurs
- Les Structures de contrôle
- Les fichiers
- Les classes
- MySQL
- Les cookies
- Les Sessions

Introduction à PHP

PHP : "Pretty Home Page" inventé par Rasmus Lerdorf en 1994

PHP : Hypertext Preprocessor

Langage de script HTML interprété coté serveur

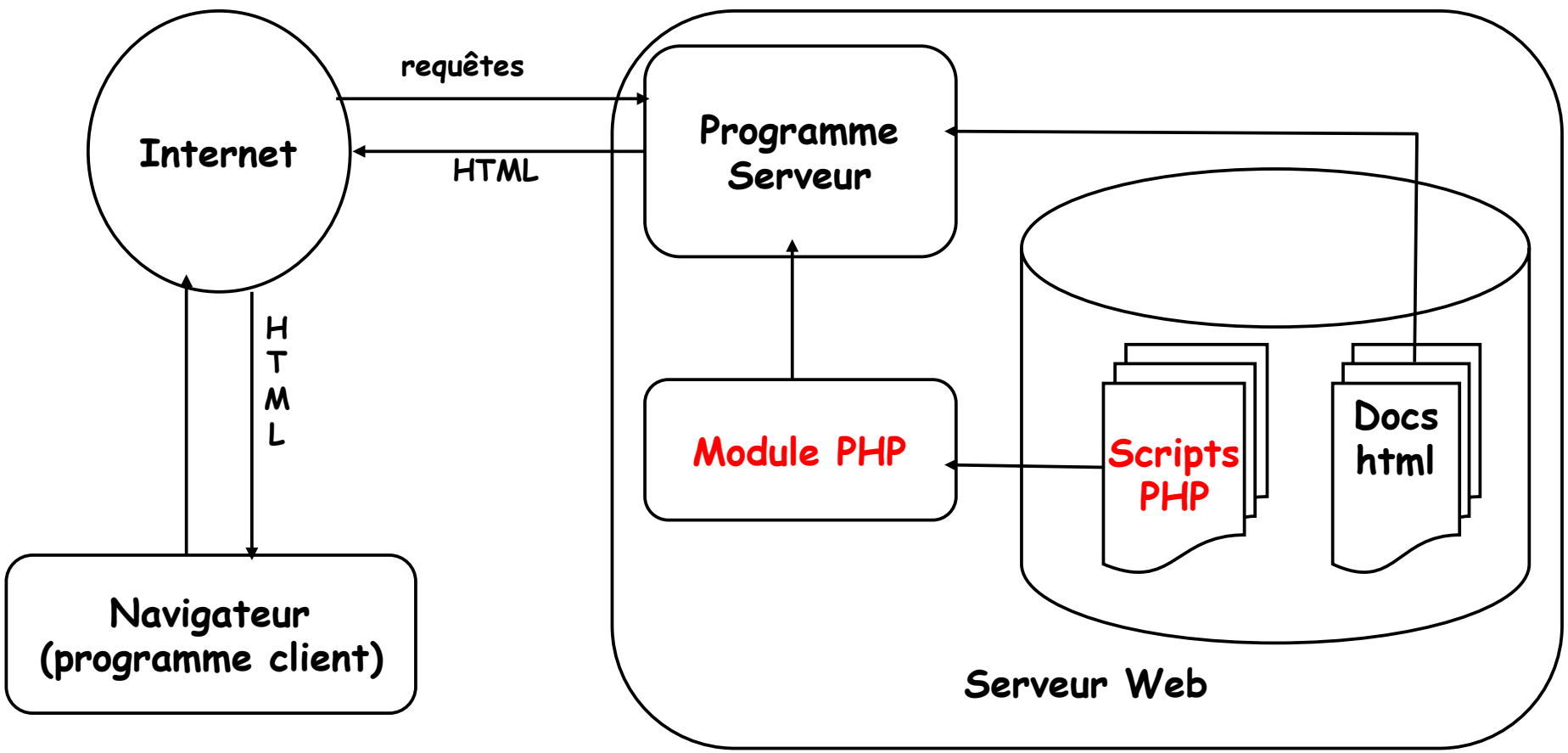
- ≠ Perl qui ne peut être insérer dans du code HTML
- ≠ Javascript qui s'exécute coté client

Soit une page HTML avec un script PHP
Soit un fichier PHP avec des lignes de HTML

Quelques sites en Français :

http://www.php.net/
http://www.nexen.net/
http://www.phpdebutant.com/

Architecture d'un site web avec PHP



Syntaxe de PHP

Généralités :

- Un fichier PHP a une extension ".php", il est toujours interprété au niveau du serveur.
- tout code entre les balises :
  1. simples : <? et ?>
  2. compatibles PHP et XML : <?php et ?> (préférées)
  3. à la Javascript : <script langage="php"> et </script>
  4. à la ASP : <% et %>
- séparateur d'instructions : « ; »
- bloc d'instructions délimité par des accolades {}
- commentaires : comme en C : « /*! » suivi de « */ »

Ou // juste sur une ligne

Syntaxe de PHP

Variables et littéraux :

- distinction minuscules/majuscules
- nom de variable toujours précédé de «! $! » et au moins un caractère non numérique
- le type de la valeur associée à une variable peut changer
- pas de déclaration de variable en PHP, c'est à la 1ère référence :

```
$mavariable = 1; echo "$mavariable";
```

Syntaxe de PHP

Constantes :

association symbole/valeur

ne peut jamais être modifié

```
define (PI,3.14159);
```

Les types :

- types numériques (entiers et flottants) et booléens

```
$i = 1;
$i = 3.14159;
$f = 0.3e-3;
```

- pas de type booléen explicite : la valeur faux est le `0`, la chaîne vide ou la chaîne `"0"`, tout le reste est vrai
- Constantes `TRUE` et `FALSE` pré-définies

Syntaxe de PHP

Les types :

- Chaînes de caractères

  entourées de «! "!» ou de «! '!»

  guillemets simples «! '!» :

  on ne peut pas mettre de variable, de caractère d'échappement, mais sauts de ligne acceptés

  Inclusion d'un «! '!» avec «! \!».

  guillemets doubles «! "!» :

  ces chaînes peuvent inclure des variables. Plus les caractères d!'échappement (\n, \r, \t, \\, \$, \")

Syntaxe de PHP

Les tableaux

- suite de valeurs référencées par une unique variable
- gestion dynamique de la taille
- tableaux multi-dimensionnels
- tableaux indicés :

  référencement par la position qui commence à 0

  - `$tab[0] = "chaîne1";`
  - `$tab[1] = 1;`

  affectation automatique d'un indice :

  - `$tab[] = "chaîne1";`
  - `$tab[] = 1;`

  instruction `array` pour initialiser un tableau

  - `$tab = array("chaîne1", 1);`

Syntaxe de PHP

Tableaux associatifs :

au lieu d'utiliser un indice, on utilise une clé, la notion d'ordre disparaît

on accède à un élément par sa clé

- initialisation :

```
$mois["Jan"] = "Janvier" ;
$mois["Fev"] = "Février";
```

- initialisation par array :

```
$mois = array ("Jan" => "Janvier" ,
               "Fev" => "Février");
```

- accès à une valeur : `$mois["Jan"];`

  donne `"Janvier"`

Syntaxe de PHP

Tableaux associatifs :

- parcours d'un tableur associatif :

```
do { echo "Clé =" . key($mois) . "Valeur =".
   current($mois) "}
while (next($mois));
```

- fonctions `next()` et `prev()`
- fonction `count()` pour compter
- fonctions pour trier : `sort()`, `ksort()`…

Syntaxe de PHP

Conversion et typage :

le type d'une variable est déterminé selon le contexte

possibilité de tester le type de la valeur référencée par une variable :

`is_string`, `is_long` (pour un entier), `is_double` (pour un flottant), `is_array`, `is_object`

fonction `gettype()` renvoie le type

possibilité de convertir le type d'une variable :

```
$v = "3 petits ânes";     ⇒ 3 petits ânes
$v = (integer) $v;        ⇒ 3
$v = (double) $v;         ⇒ 3
```

Syntaxe de PHP

Les opérateurs :

- opérateurs arithmétiques (+, -, *, /, %)
- concaténation de chaîne par «! .!»
- incrémentation avec `$i++` ou `++$i` (ou décrémentation)
- opérateurs de bits (& (ET binaire), | (OU binaire), ^ (OU exclusif), inversion et décalages)
- opérateurs logiques (&& (ou and), || (ou or), xor et !)
- opérateurs de comparaison (= =, !=, <, >, <= et >=)

Syntaxe de PHP

Les structures de contrôle :

- mêmes constructions qu'en C
- les test :
  - if-else
  - switch
- les Boucles :
  - while
  - do-while
  - for

Syntaxe de PHP

Les expressions :

tout ce qui produit une valeur

Les instructions break et continue :

`break` déclenche la sortie forcée de la boucle

`continue` dirige l'exécution à la prochaine évaluation du test de continuation en sautant les éventuelles instructions complétant le corps de la boucle

Syntaxe de PHP

Les structures de contrôle : test

Le if… else :

```
if (expression)
{…}
else {…}
```

Le switch :

```
switch (expression)
{  case valeur1 :    …
                     break;
   case valeur2 :    …
                     break;
   default :         …
}
```

Syntaxe de PHP

Les structures de contrôle : boucles

Le **while** :

```
while (expression){
   actions;
}
```

Le **do-while** :

```
do {
   actions;
} while (expression);
```

Le **for** :

```
for (expression1; expression2; expression3){
   actions;
}
```

Syntaxe de PHP

Les fonctions :

- les fonctions doivent être définies avant leur appel
- syntaxe :

```
function Nom ([$arg1, $arg2, …. ])
    { …. }
```

- renvoi d'une seule valeur avec `return`
- passage des arguments par valeur par défaut
- on indique un passage par adresse avec «! &! »
- 3 types de variables :
  - variable automatique par défaut : `$variable = 0;`
  - variable statique (persistantes entre 2 appels) :

```
static $variable = 0;
```

  - variable globale visible partout : `global $variable = 0;`

Syntaxe de PHP

Les variables d'environnement :

- **QUERY_STRING : récupère l'ensemble des informations transmises depuis un formulaire (visibles au niveau du navigateur en utilisant la méthode GET)**

- **REQUEST_METHOD : le résultat est GET ou POST**

- **HTTP_USER_AGENT : permet d'avoir des informations sur le type de navigateur utilisé par le client, ainsi que son système d'exploitation**

La récupération se fait grace à $_SERVER['variable_environnement']

- **$_POST ou $_GET : tableau associatif global contenant les données transmises exemple :** $_GET['var']

Création de pages web

Fonctions permettant d'envoyer du texte au navigateur :

La fonction echo :

permet d'envoyer au navigateur la chaîne de caractères ou une expression que l'interpréteur évalue

```
echo "Bonsoir à tous";
echo (1+2)*87;
```

La fonction print :

similaire à la fonction echo, l'expression à afficher est entre parenthèses : `print(expression);`

```
print("Bonsoir à tous");
print ((1+2)*87);
```

La fonction printf :

celle du langage C, rarement utilisée, permet un formatage des données affichée à l'écran

Syntaxe de PHP

Les Fichiers :

- Par le nom relatif : `"../../images/cnam.gif"`
- Par le nom absolu : `"/users/work/f-yv/file.txt"`
- Par une URL : `http://www.cnam.fr/index.html`

Les commandes :

- `require(string nom_fichier)` : cette commande se remplace dans le script toujours par le contenu du fichier
- `include(string nom_fichier)` : inclus et évalue le fichier spécifié en argument

Syntaxe de PHP

Les tests de fichiers :

La fonction is_file permet de savoir si le nom est passé en paramètre correspond à un fichier existant (ni lien symbolique, ni répertoire)

```
booléen is_file(chaine Nom_du_fichier);
```

renvoie 1 si il s'agit d'un fichier, 0 dans le cas contraire

La fonction is_dir permet de savoir si le fichier dont le nom est passé en paramètre correspond à un répertoire

```
booléen is_dir(chaine Nom);
```

renvoie 1 si il s'agit d'un répertoire, 0 dans le cas contraire

La fonction is_executable permet de savoir si le fichier dont le nom est passé en paramètre est exécutable

```
booléen is_executable(chaine Nom_du_fichier);
```

renvoie 1 si le fichier est exécutable, 0 dans le cas contraire

ptr opendir (chaine Nom_du_dossier)	Pointe sur les éléments du dossier
file readdir (pointeur ptr)	Récupère l'élément courant

Syntaxe de PHP

Ouvrir un fichier :

```
int fopen(string nom_fichier, string "mode")
```

Modes possibles :

r : ouverture en lecture seulement

w : ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)

a : ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

r+ : ouverture en lecture et écriture

w+ : ouverture en lecture et écriture (la fonction crée le fichier s'il n'existe pas)

a+ : ouverture en lecture et écriture avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

Fermer un fichier :

Un fichier ouvert avec la fonction fopen doit être fermé par la fonction fclose en lui passant en paramètre "int" retourné par la fonction fopen

```
bool fclose ( int nom_fichier)
```

Syntaxe de PHP

Lire dans un fichier :

```
string fgets ( int fichier, int length);
```

retourne la chaîne lue jusqu'à la longueur length - 1 octet, ou bien la fin du fichier, ou encore un retour chariot (le premier des trois qui sera rencontré)

```
array file ( string filename );
```

retourne le fichier dans un tableau. Chaque élément du tableau correspond à une ligne du fichier, et les retour-chariots sont placés en fin de ligne

Ecrire dans un fichier :

```
int fwrite ( int fichier, string string [, int length]);
```

écrit le contenu de la chaîne "string" dans le fichier "fichier". Si la longueur "length" est fournie, l'écriture s'arrêtera après length equivalent à fputs

Manipuler un fichier :

bool file_exists (int fichier) ;	True si le fichier existe
bool feof (int fichier) ;	True si à la fin du fichier
array explode (string symbole, string str) ;	Tableau délimité par symbole
bool empty(string str) ;	True si str vide

Les classes et les objets

Un objet a une existence physique (il peut être affecté à une variable)

Une classe peut être vue comme une abstraction de tous les éléments communs à un ensemble d'objets (en structure et en comportement)

Une classe est une déclaration des propriétés et des méthodes :

- Propriété = variable de classe
- Méthode = fonction de classe

L'environnement de la classe est protégé :

- Les propriétés et les méthodes peuvent être déclarées comme non accessibles en dehors de la classe
- Les propriétés ne se mélangent pas aux variables globales

Les classes et les objets

Un objet est une instance d'une seule et unique classe

Depuis PHP 5, les objets sont des références

```
$objet1 = $objet2;
```

ne duplique pas l'objet $objet1, mais crée une deuxième référence vers le même objet $objet1

Une classe permet de créer des objets au moyen d'un "constructeur" appelé, par défaut, "new" suivi du nom de la classe

=> PHP créé l'objet en mémoire et référence son emplacement en mémoire dans la variable :

```
$object = new MaClasse();
```

La variable $object contient une référence à un objet de la classe MaClasse, mais pas l'objet lui-même

Pour détruire cet objet en mémoire, il faut détruire toutes les références vers cette instance de la classe :

```
unset($object);
```

toutes les références sont détruites, l'objet n'existe donc plus en mémoire

Les classes et les objets

La visibilité d'une propriété ou d'une méthode est définie en préfixant la déclaration avec un mot-clé :

- `public`
- `protected`
- `Private`

Les éléments déclarés publics (`public`) peuvent être utilisés par n'importe quelle partie du programme

L'accès aux éléments protégés (`protected`) est limité aux instances de la classe et des sous classes de cette classe

L'accès aux éléments privés (`Private`) est uniquement réservé aux instances de la classe qui les a définis

Les classes et les objets

Une classe (mot clé : `class`) est la définition d'un ensemble d'objets et est composée de :

- attributs : (mot clé : `public/protected/private`) données caractérisant l'état de l'objet
- méthodes : (mot clé : `fonction`, précédé du mot clé : `public/protected/private`) opérations applicables aux objets

En absence de déclaration de visibilité, l'attribut ou la méthode est considéré comme `public` (compatibilité avec PHP 4)

Un attribut ou une méthode déclaré `static` permet d'accéder à l'attribut ou la méthode sans instanciation de la classe

Les classes et les objets

```
<?php
    class nouvelle_classe {
    private $atr1; private $atr2; private $atr3;
    public function fonct1() {
    …}};

    //création d'un objet
    $objet1 = new nouvelle_classe();

    //affectation des propriétés de l'objet
    $objet1->atr1=…; $objet1->$atr2=…; $objet1->$atr3=…;

    //utilisation des propriétés
    echo "l'attribut de l'objet1 est ", $objet1->atr1, "<br />";
    //appel de la méthode fonct1()
    $objet1->fonct1() ;
?>
```

Les classes et les objets

Manipulation des classes et des objets

- Instanciation de la classe : Constructeur (par défaut) de même nom que la classe et appelé par l'opérateur **new**

```
$Nom_de_l_objet = new Nom_de_la_classe;
```

- Accéder aux propriétés d'un objet

```
$Nom_de_l_objet->Nom_de_la_donnee_membre = Valeur;
```

- Accéder aux méthodes d'un objet

```
$Nom_de_l_objet->Nom_de_la_fonction_membre(par1,par2,…);
```

- La variable **$this** permet d'appeler à un objet de s'appeler lui-même :

```
$this->atr1 = $atr1;
```

Les classes et les objets

Héritage :

- **Instruction `extends`**

  Exemple : `class Camion extends Vehicule`

La classe Camion hérite des attributs et des méthodes appartenant a la classe Vehicule et possède ses propres attributs et méthodes

- **pas d'héritage multiple**

```php
class Camion extends Vehicule
{
    $modele;
    function  accelerer()
    {
      $this->vitesse = $this->vitesse + 5;
    }
}
```

```php
class Vehicule {
   public $name;
   public $color;
   private $prix;
   public $vitesse=0;
   function Get_price()
      {return $this->prix;}
}
```

- **Dans cette exemple l'objet Camion contient tous les attributs et methodes de Vehicule ainsi que ses propres attributs**

Les classes et les objets

Sauvegarde des objets :

- **serialize** transforme un objet en une chaîne de caractères
- **unserialize** reconstitue l'objet à partir de la chaîne de caractères enregistrée

Exemple :

```
<?php // Page1 : page1.php
     include("classe.inc"); // inclusion de la définition de classe
     $objet1 = new classe();
               // crée une instance de l'objet
     $objet_chaine = serialize($objet1); // sérialise l'objet
     $fichier = fopen("fichier1", "w");
               // ouvre fichier fichier1 en écriture seule
     fputs($fichier, $objet_chaine);
               // écrit l'objet sérialisé dans le fichier fichier1
     fclose($fichier); // ferme le fichier fichier1
?>
```

Les classes et les objets

Sauvegarde des objets :

```
<?php // Page2 : page2.php
    include("classe.inc");
            // inclusion de la définition de classe
      /* regroupe tous les éléments du tableau  retourné par la
 fonction file dans une chaîne */
  $objet_chaine = implode("", file("fichier1"));
  $objet1 = unserialize($objet_chaine); // désérialise l'objet
      // appelle les attributs de l'objet
  $objet1->atr1 = …;
  $objet1->atr2 = …;
  $objet1->atr3 = …;
?>
```

Serialisation : serialize($obj)

Désirialisation : unserialize($chaine)

Les classes et les objets

Fonctions sur les classes et les objets :

`get_class()`	Retourne la classe d'un objet
`get_parent_class()`	Retourne les super-classes d'un objet
`method_exists()`	Vérifie si la méthode existe dans un objet
`class_exists()`	Vérifie si la classe existe
`is_subclass_of()`	Vérifie si une classe est une sous classe d'une autre
`get_class_methods()`	Retourne les méthodes d'une classe dans un tableau
`get_declared_classes()`	Retourne les classes déclarées dans un tableau
`get_class_vars()`	Retourne les variables de classe dans un tableau
`get_object_vars()`	Retourne les variables d'un objet dans un tableau

Abstraction de classes

Une méthode abstraite n'a pas de code (juste la déclaration de la signature de la méthode, mais pas d'implantation)

Une classe abstraite n'a pas d'instance

La déclaration d'une classe abstraite est précédée du mot clé "abstract"

Toutes les classes contenant au moins une méthode abstraite doivent également être déclarées abstraites

La "surcharge" en PHP est différente de celle des autres langages orientés objet

La surcharge en PHP permet de créer dynamiquement, par des méthodes magiques, des membres et des méthodes d'une classe

Les méthodes surchargées doivent être définies comme "public"

Depuis PHP 5, une méthode préfixée par le mot-clé "final" ne peut pas être surchargée dans une classe filles

Si la classe est définie comme "final", elle ne peut pas être étendue

Les méthodes magiques

Les **méthodes magiques** sont des méthodes qui, si elles sont déclarées dans une classe, ont une fonction déjà prévue par le langage; elles sont appelées automatiquement :

`_construct()` : Constructeur de la classe

`_destruct()` : Destructeur de la classe

`_set()` : Déclenchée lors de l'accès en écriture une propriété de l'objet

`_get()` : Déclenchée lors de l'accès en lecture 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, depuis PHP 5.3)

`_isset()` : Déclenchée si on applique `isset()` une propriété de l'objet

`_unset()` : Déclenchée si on applique `unset()` une propriété de l'objet

`_sleep()` : Exécutée si la fonction `serialize()` est appliquée à l'objet

`_wakeup()` : Exécutée si la fonction `unserialize()` est appliquée à l'objet

`_toString()` : Appelée lorsque l'on essaie d'afficher directement à l'objet :

```
echo $object;
```

`_set_state()` : Méthode statique, lancée lorsque l'on applique la fonction `var_export()` à l'objet

`_clone()` : Appel 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

Clonage d'objets

La méthode __clone() d'un objet ne peut être appelée directement : le mot-clé clone (qui fait appel à la méthode __clone() de l'objet, si elle a été définie dans la classe)

```
<?php
        $objet_copie = clone $objet;
?>
```

Dans l'objet cloné, $objet_copie, tous les attributs (propriétés) qui sont des références à d'autres variables restent des références

```
        $objet1 == $objet2
```

Les objets $objet1 et $objet2 sont égaux si et seulement ils sont des instances de la même classe et s'ils ont les mêmes valeurs d'attributs

```
        $objet1 === $objet2
```

Les objets $objet1 et $objet2 sont identiques si et seulement ils font référence à la même instance de la même classe

```
        $objet_copie == $objet (égaux)
mais    $objet_copie !=== $objet (mais pas identiques)
```

Interfaces

Pour éviter l'héritage multiple, les langages orientés-objet utilisent les interfaces d'objet de la classe pour spécifier quelles méthodes une classe doit implanter de l'autre classe :

- Une interface est définie par le mot-clé `interface`
- Une interface n'a pas de propriété
- Une interface n'a pas que des noms de méthodes publiques

Pour définir une classe qui implante des interfaces doit les déclarer par

```
Implements interface1, interface2, …, interfaceN
```

- Toutes les méthodes de ces interfaces doivent être définies dans des classes
- Pour éviter toute ambiguïté, les divers noms de méthodes doivent différents

Interfaces

Exemple d'interface :

```php
interface SpeedRegulator
{
    public function accelerer();
    public function freiner();
}
```

```php
class Voiture implements SpeedRegulator
{
    public $name;
    public $vitesse=0;
    function accelerer()
    {
        if($this->vitesse + 10 < 120)
        $this->vitesse = $this->vitesse + 10;
    }
    function freiner()
    {
        if($this->vitesse - 10 > 0)
        $this->vitesse =  $this->vitesse -10;
    }
}
```

```php
class Camion implements SpeedRegulator
{
    public $name;
    public $vitesse=0;
    function  accelerer()
    {
        if($this->vitesse + 5 < 100)
        $this->vitesse = $this->vitesse + 5;
    }
    function freiner()
    {
        if($this->vitesse - 5 > 0)
        $this->vitesse =  $this->vitesse - 5;
    }
}
```

Les constantes magiques

Les constantes magiques donnent des informations sur :

__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 PHP 5.3)

__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 PHP 5.3)

MySQL

MySQL est un SGBD relationnel

- Gratuit (www.mysql.com), sous la licence GNU GPL
- Disponible sous Windows et tous les Unix (Mac OSX)
- Propose le minimum vital :

  (Presque) tout le langage SQL

  Stockage, indexation, optimisation

  Securité (pannes)

  Gestion de droits d'accès (multiples en simultané)

- Qualités reconnues (outre la gratuité) :

  Simplicité

  Efficacité

  Robustesse

MySQL

Quelques limites de MySQL (standard):

- Pas de transactions dans la version standard :

  Inadapté pour les applications transactionnelles (finances, réservations…)

  Ni commit, ni rollback…

  Reprise sur panne rudimentaire

- Pas de requêtes imbriquées (pas très grave ?)

- Pas d'environnement de développement

  Pas de générateur d'écran ou de rapport

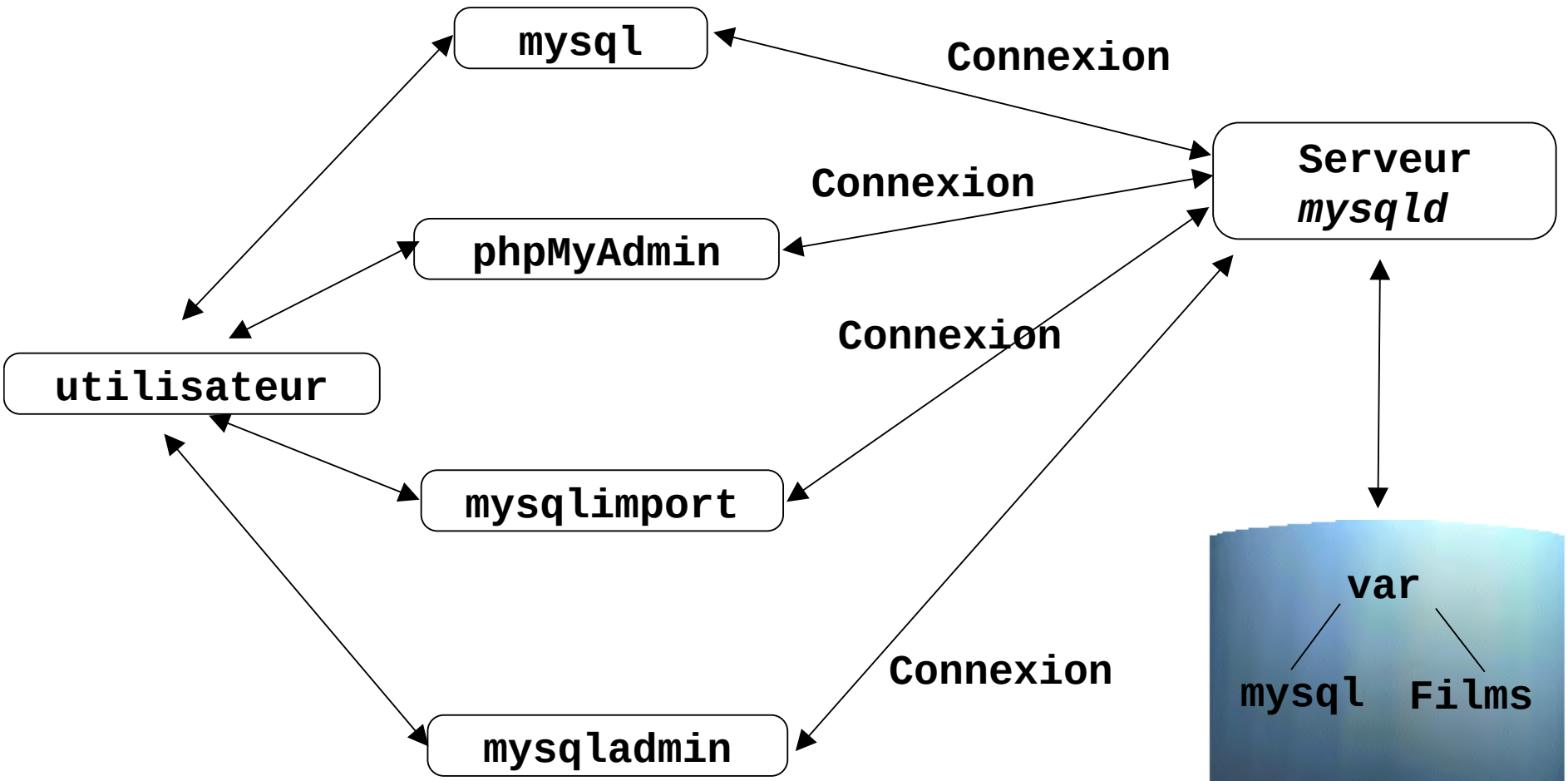
  Pas d'outil pour le DBA (juste une API en C !)…

MySQL

Statut de MySQL :

- Très prisé pour les sites web du (très bonne intégration avec Apache et PHP)
- En constant développement
- Beaucoup de "contributions" extérieures
  - Des API en C++, Perl, PHP, Java (JDBC)…
  - Des clients graphiques, des utilitaires…
- MySQL est resté simple et efficace
- Soutenu par SAP, Intel…

MySQL



Architecture de MySQL

MySQL

Le serveur `mysqld` est en écoute sur un port réseau (le 3306 par défaut)

Il gère une ou plusieurs bases de données

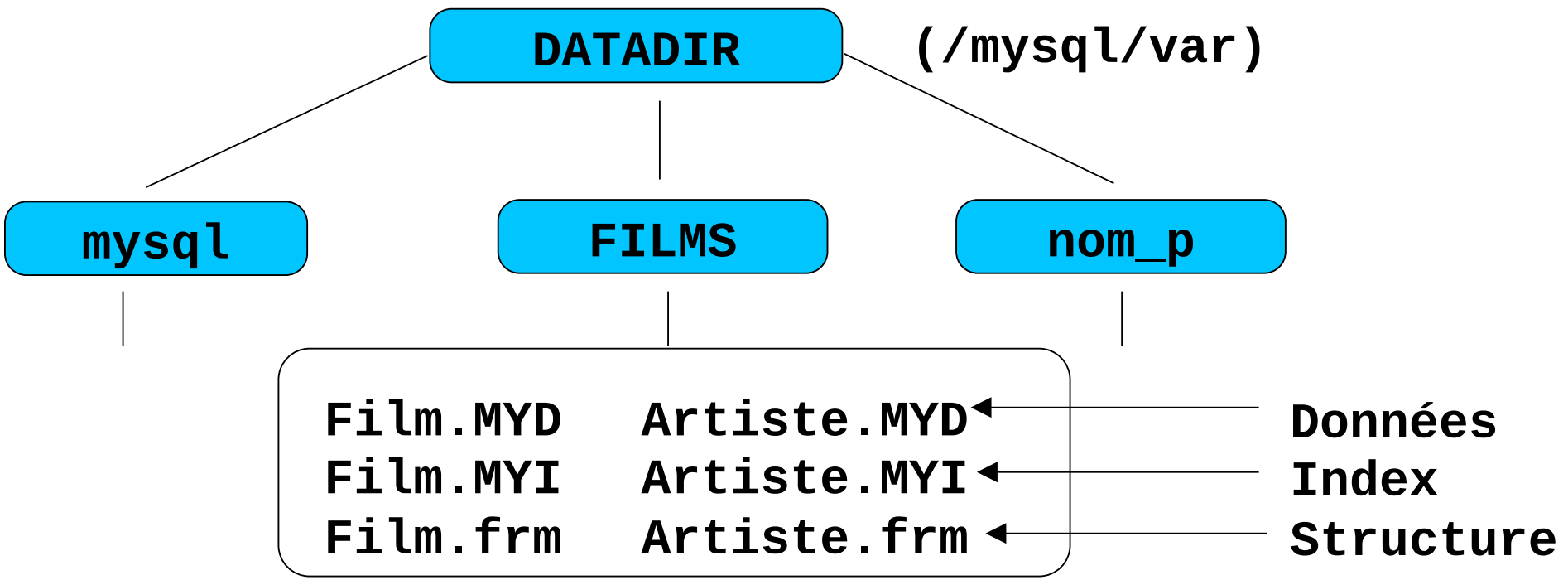
- La base `mysql` est le "dictionnaire de données"
- Les autres bases sont des bases utilisateur

Il dialogue avec ses clients en multi-threading (processus léger)

MySQL

Les bases de données MySQL :

- Une base = un répertoire !
- Une table = 3 fichiers !



MySQL

Le client `mysql` permet de se connecter sous un nom d'utilisateur:

```
mysql -u root -p Films
```

=> se connecte sous `root` (MySQL) à la base Films, avec un mot de passe (p)

Permet d'entrer toutes les commandes (en fonction des droits d'accès bien sûr)

MySQL

Un utilisateur MySQL est défini par :

- son nom
- le nom (Internet) de sa machine

Exemple :

User : Root
Machine : Localhost

Creer un utilisateur :

```
CREATE USER 'visiteur'@'localhost' IDENTIFIED BY 'mdpvisiteur';
```

MySQL

Un utilisateur peut avoir plusieurs types de droits:

- sur le serveur (administration)
- sur les bases et tables (création, destruction)
- sur les données (lecture, écriture)
- sur d'autres utilisateurs (transmission de droits)

Le compte root est le DBA: il a tous les droits sur tout

MySQL

La commande GRANT est utilisée pour créer un utilisateur avec tous les droits sur la base Films :

```
GRANT ALL ON Films.*
TO saad@localhost
IDENTIFIED BY 'mdpsaad'
```

Pour créer un utilisateur avec des droits en lecture sur la table Artiste:

```
GRANT select ON Films.Artiste
TO visiteur@localhost
IDENTIFIED BY 'mdpVisiteur'
```

MySQL

Création de bases et de tables sous root :

```
mysql> CREATE DATABASE Films;
```

Tout utilisateur ayant des droits peut créer des tables dans une base :

```
mysql> CREATE TABLE  Film (...);
```

Les commandes SQL **CREATE**, **ALTER**, **DROP** sont reconnues

MySQL

MySQL reconnaît les principaux types SQL, et quelques autres :

- `INTEGER` : les entiers
- `FLOAT` : numériques flottants
- `DECIMAL (M, D)` : numériques exacts
- `CHAR` : chaînes de longueur fixe
- `VARCHAR` :chaînes de longueur variable
- `TEXT` : textes longs
- `DATE` : dates
- `TIME` : moments

MySQL

Exemple de création de table :

```
CREATE TABLE Film
      (id INT NOT NULL AUTO_INCREMENT
       titre          VARCHAR (50) NOT NULL,
       annee     INTEGER NOT NULL,
       idMES     INTEGER,
       resume        TEXT,
       codePays     VARCHAR (4),
       PRIMARY KEY (titre),
       FOREIGN KEY (idMES) REFERENCES Artiste,
       FOREIGN KEY (codePays) REFERENCES Pays);
```

MySQL

Insertion dans une table :

```
mysql> CREATE TABLE MaTable
   -> (MaDonnee INTEGER NOT NULL,
     -> PRIMARY KEY (MaDonnee));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO MaTable (MaDonnee) VALUES (1);
Query OK, 1 row affected (0.05 sec)
```

Il vaut mieux placer toutes les commandes dans un fichier de script : MonF.sql

```
mysql -u MonNom -p MaBase < MonF.sql
```

Exemple d'un fichier de script :

```
# Création d'une table 'FilmSimple'
USE Films;
CREATE TABLE FilmSimple
  (titre       VARCHAR (30),
   annee       INTEGER,
   nomMES      VARCHAR (30),
   prenomMES   VARCHAR (30),
   anneeNaiss INTEGER
  );
```

Plusieurs connexion php

Plusieurs connexions php simultanées :

```
<?php
  $connexion1 = mysql_pconnect (serveur1, nom1, pass1);
  mysql_select_db (base1, $connexion1);
  $connexion2 = mysql_pconnect (serveur2, nom2, pass2);
  mysql_select_db (base2, $connexion2);

  $resultat1 = mysql_query ($requete1, $connexion1);
  $resultat2 = mysql_query ($requete2, $connexion2);

  traitement($resultat1,$resultat2);

?>
```

MySQL

Bases autre que MySQL -> interface commune : ODBC (Open Database Connectivity)

Fonctions php valable pour tout SGBD

- odbc_connect(db,login,pass);
- odbc_exec(requete,connexion);
- odbc_fetch_row(connexion);

Exemple :

```
<?php
 $connexion1 = odbc_connect (“mysql:localhost”,...);
 $connexion2 = odbc_connect
      (“oracle:db.fournisseur.fr”...);
 $connexion3 = odbc_connect
      (“postgres:128.176.212.3”,...);
?>
```

MySQL

MySQL en résumé :

- Tout à fait conforme à la norme SQL ANSI

  CREATE TABLE, DROP TABLE, ALTER TABLE

  SELECT, INSERT, DELETE, UPDATE

- Avec des extensions (très utiles)

  Types de données (TEXT, BLOB)

  Contraintes (AUTO_INCREMENT)

  Beaucoup de fonctions

MySQL/PHP

Création de documents web à partir d'une BD => application avec MySQL/PHP

Architecture client/serveur à trois tiers :

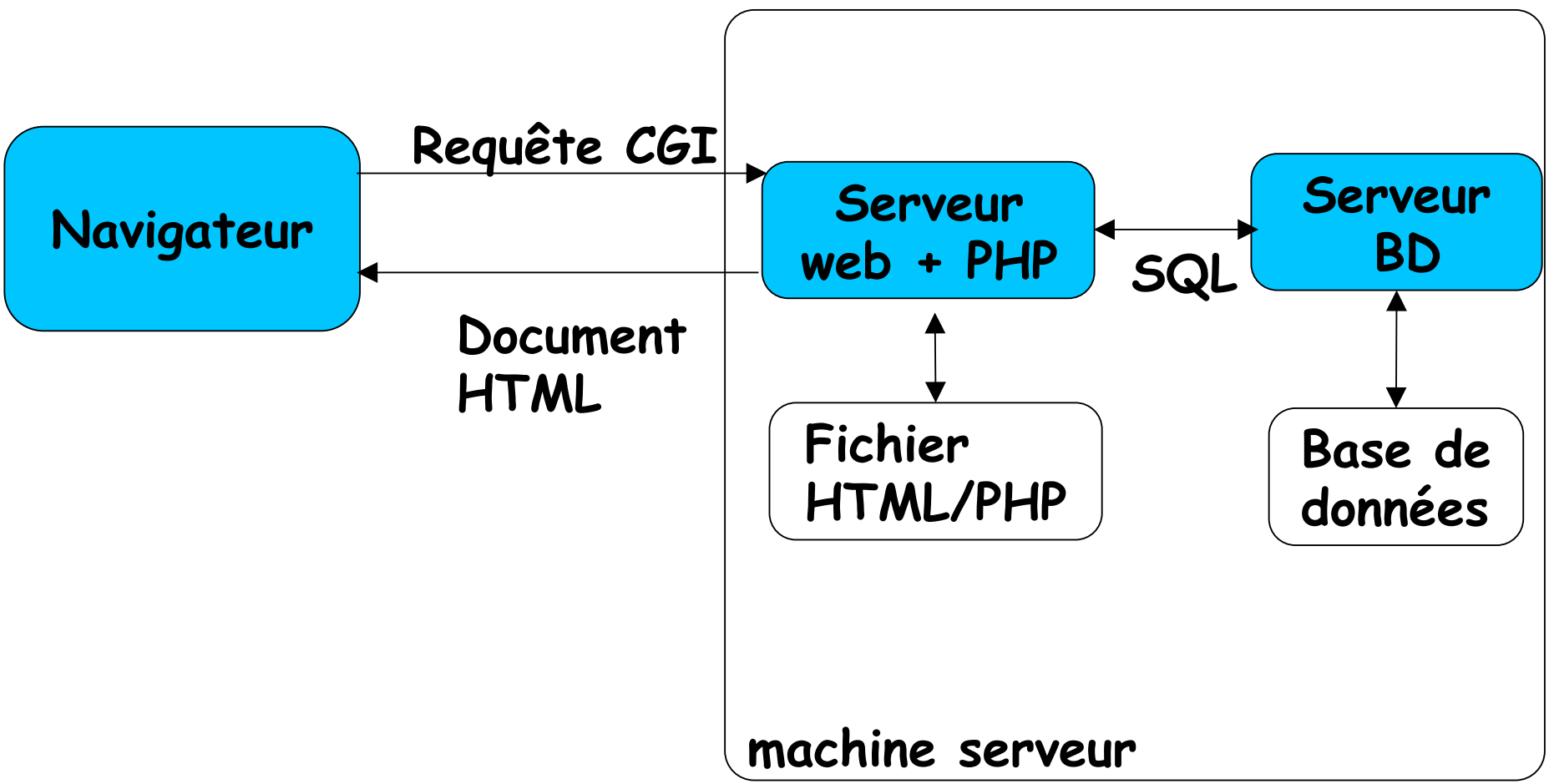
- MySQL :

  stockage, de la protection des données, de l'interface SQL
- PHP :

  extrait des données et les met en forme

  reçoit des données et les stocke
- Le navigateur fournit l'interface graphique

Architecture Web avec base de données



MySQL/PHP

Connexion à MySQL avec un script PHP :

- `mysql_pconnect (serveur, nom, pass)`
  établit une connexion
  si OK, renvoie un identifiant non nul cnx
- `mysql_select_db(base, cnx)`
  se place dans une base et renvoie vrai si OK
- `mysql_query (requete, cnx)`
  exécute une requête et renvoie un identifiant
- `mysql_fetch_object (resultat)`
  renvoie la ligne suivante sous forme d'objet

MySQL/PHP

Interrogation et affichage (exemple) :

```
<?php
  $connexion = mysql_pconnect ("localhost",
     "MonNom", "MonMDP");

if (!$connexion)
  {
    echo "Désolé, connexion impossible\n";
    exit;
  }

  if (!mysql_select_db ("MaBase", $connexion))
  {
    echo "Désolé, accès à la base impossible\n";
    exit;
  }
```

MySQL/PHP

Interrogation et affichage (exemple) :

```
$resultat = mysql_query ("SELECT * FROM FilmSimple",
      $connexion);
if ($resultat)
   while ($film = mysql_fetch_object ($resultat))
       echo "$film->titre, paru en $film->annee, réalisé "
            ."par  $film->prenomMES $film->nomMES.<BR>\n";
  else
  {
   echo "<B>Erreur dans l'exécution de larequête.</B><BR>";
   echo "<B>Message :</B> " .mysql_error($connexion);
  }
```

MySQL/PHP

En association avec un formulaire :

```
<H1>Formulaire pour programme PHP</H1>

<FORM ACTION='ExMyPHP2.php' METHOD=POST>
  <P>
  Titre :  <INPUT TYPE=TEXT SIZE=20 NAME = 'titre'> <P>
  Année début : <INPUT TYPE=TEXT SIZE=4 NAME='anMin'>
  Année fin : <INPUT TYPE=TEXT SIZE=4 NAME='anMax'> <P>
  <P>
   <B>Comment combiner ces critères. </B>
      ET <INPUT TYPE=RADIO NAME='comb' VALUE='ET'>
     OU <INPUT TYPE=RADIO NAME='comb' VALUE='OU'> ?
 <INPUT TYPE=SUBMIT VALUE='Rechercher'>
</FORM>
```

MySQL/PHP

On récupère donc les variables $titre, $anMin, $anMax, et $comb

```
<?php
  echo "<B>Titre = $titre anMin = $anMin anMax= $anMax\n";
  echo "Combinaison logique : $comb<P></B>\n";

  if ($comb == 'ET')
    $requete = "SELECT * FROM FilmSimple "
          . "WHERE titre LIKE '$titre' "
          . "AND annee BETWEEN $anMin and $anMax";
  else
    $requete = "SELECT * FROM FilmSimple "
          . "WHERE titre LIKE '$titre' "
          . "OR (annee BETWEEN $anMin and $anMax)";
```

MySQL/PHP

Exécution de la requête :

```
<?php
   $connexion = mysql_pconnect (SERVEUR, NOM, PASS);

  mysql_select_db (BASE, $connexion);
  $resultat = mysql_query ($requete, $connexion);
  while ( ($film = mysql_fetch_object ($resultat)))
       echo "$film->titre, paru en $film->annee, réalisé "
          . "par  $film->prenomMES $film->nomMES.<BR>\n";
?>
```

MySQL/PHP

Mises à jour de la base :

- Un formulaire de saisie
- La fonction `mysql_query` permet d'exécuter l'ordre! :

  `INSERT` pour des insertions

  `UPDATE` pour une modification

  `DELETE` pour une destruction

MySQL/PHP

Exemple: mise à jour de la table FilmSimple :

```
<FORM  ACTION="ExMyPHP3.php"  METHOD=POST>
  Titre :  <INPUT TYPE=TEXT SIZE=20 NAME="titre"><BR>
  Annee :  <INPUT TYPE=TEXT SIZE=4  NAME="annee"> <P>
  Nom :    <INPUT TYPE=TEXT SIZE=20 NAME="prenom"> <BR>
  Prenom : <INPUT TYPE=TEXT SIZE=20 NAME="nom"> <BR>
  Annee  : <INPUT TYPE=TEXT SIZE=4  NAME="anneeNaissance">

  <H1>Votre choix</H1>
  <INPUT TYPE=SUBMIT VALUE='Inserer' NAME='inserer' >
  <INPUT TYPE=SUBMIT VALUE='Modifier' NAME='modifier' >
  <INPUT TYPE=SUBMIT VALUE='Detruire' NAME='detruire' >
</FORM>
```

MySQL/PHP

L'action dépend du bouton choisi par l'utilisateur :

```
// Test du type de mise à jour effectuée

  echo "<HR><H2>\n";
   if (isset($inserer)) echo "Insertion du film $titre";
   elseif (isset($modifier)) echo "Modification du film $titre";
   elseif (isset($detruire)) echo "Destruction du film $titre";
   echo "</H2><HR>\n";

// Affichage des données du formulaire

  echo "Titre: $titre <BR> annee: $annee<BR>\n";
   echo "Mis en scène par $prenom $nom, né en $anneeNaiss\n";
```

MySQL/PHP

Choix de la requête en fonction de l'action demandée :

```
if (isset($inserer))
    $requete = "INSERT INTO FilmSimple (titre, annee, "
             . "prenomMES, nomMES, anneeNaiss) "
             . "VALUES ('$titre', $annee, "
             . "'$nom', '$prenom', $anneeNaiss) ";
if (isset($modifier))
    $requete = "UPDATE FilmSimple SET annee=$annee, "
        . "prenomMES = '$prenom', nomMES='$nom', "
        . "anneeNaiss=$anneeNaiss WHERE titre = '$titre' ";
if (isset($detruire))
    $requete = "DELETE FROM FilmSimple WHERE titre='$titre'";

$resultat = mysql_query ($requete, $connexion);
```

MySQL/PHP

Informations sur le schéma, étant donné le résultat (`$res`) d'une requête :

- `mysql_num_fields($res)` donne le nombre d'attributs
- `mysql_field_name($res, $i)` donne le nom de l'attribut
- `mysql_fetch_row ($res)` renvoie une ligne du résultat sous la forme d'un tableau indicé

Cookies & sessions

Le serveur ne distingue pas les connexions successives

Problèmes :

- Comment identifier un utilisateur ?
- Comment faire pour que les données relatives à un utilisateur soit disponibles pour tous les scripts du site web ?

Utilisation des cookies :

l'utilisateur saisit (login,password)

le navigateur enregistre les cookies (login,password)

chaque page du site teste l'existence de ces cookies

-> identification de l!'utilisateur

Cookies & sessions

Données associées à l'utilisateur (taille, sécurité) dans la base de données (impossible dans les cookies)

- Identification de l'utilisateur -> numéro de session unique
- cookies (login, passwd, numero_session)

Login	session	limite	donnée1	donnée2 …
dgram	552	5mn	rateau	3
Moi	127	2mn	pelle	1

…

Chaque script retrouve les données associées grâce au numéro de session

Cookies & sessions

Un cookie est un fichier texte créé par un script et stocké sur l'ordinateur des visiteurs d'un site

Les cookies étant stockés sur le poste client, l'identification est immédiate et ne peuvent apparaître sur l'écran d'un visiteur que les renseignements qui le concernent

Pour des raisons de sécurité, les cookies ne peuvent être lus que par des pages issues du serveur qui les a créés

Le protocole HTTP étant sans état, il ne conserve pas les informations transmises par le même utilisateur

Netscape a étendu le protocole HTTP avec deux entêtes :

`Set-Cookie` : envoyé par le serveur pour créer un cookie (un `Set-Sookie` par cookie)

`Cookie` : envoyé par le client (un `Cookie` peut envoyer plusieurs cookies)

Cookies & sessions

L'en-tête HTTP Set-Cookie comprend :

- nom=valeur : définition du cookie
- expires : date de fin de validité
- domain : le domaine pour lequel le cookie est visible
  Par défaut : nom du serveur qui créé le cookie
- (path,secure,...)
  Secure : 1 pour une connexion sécurisée, 0 sinon

Tout est optionnel, à l'exception de la définition du cookie

```
Set-Cookie:
MonCookie=200;
expires=Mon,24-Dec-2004 12:00:00 GMT;
domain=cnam.fr
```

Cookies & sessions

PHP a 2 fonctions pour gérer les cookies :

```
setcookie(nom,valeur[, date d'expiration en timestamp])
setrawcookie()
```

Exemple d'en-tête avec un `set-cookie` (résultat obtenu par telnet) :

```
HTTP/1.1 200 OK
Date: Mon, 27 Oct 2008 09:11:54 GMT
Server: Apache/2.0.55 (MacOSX) PHP/5.1.2
X-Powered-By: PHP/5.1.2
Set-Cookie: id=5
Content-Length: 0
Content-Type: text/html
```

`setrawcookie()` est exactement la même que `setcookie()` excepté que la valeur du cookie ne sera pas automatiquement encodée selon la règle des URLs lors de l'envoi au navigateur (d'où le "raw")

Cookies & sessions

Création des cookies : fonction **setcookie()**

- fonction à placer au début du script

```
Setcookie("nom_var", "valeur_var", date_expiration,
 "chemin", "domain", " secure")
```

```
<?php
setcookie (“MonCookie", “Bonjour", time()
 +3600*24*60) ;
if (!$MonCookie) {
          echo “le cookie n’a pas été défini”;}
     else {
          echo $MonCookie, “<br>”;
}
?>
```

Cookies & sessions

Pour spécifier un domaine de visibilité :

```
//Le cookie ne sera visible que sur page.php sur le domaine
 monsite.com
setcookie('id',5,null,'/page.php','monsite.com');

//Le cookie ne sera visible que sur page.php sur le domaine monsite.com
setcookie('id',5,null,'/page.php','monsite.com','httponly');
//httponly permet d'empêcher la lecture du cookie par javascript par
 exemple
```

Le nombre de cookies est limité par les navigateurs (mais des centaines pour firefox)

La taille d'un cookie doit être inférieure à 5KO

La sécurité des cookies est limité (sensibilité)

Un cookie est "sécurisé" si et seulement si la connexion se fait via une connexion SSL

Cookies & sessions

Lecture de cookies

- Vérification de l'existence des variables dont les noms et les valeurs ont été définis lors de la création du cookie : fonction `isset($nom_var)`
- `isset($nom_var)` renvoie `true` si la variable `$nom_var` existe et `false` sinon

```
<?php
if (isset ($nom_var)) {
          echo "<h2> Bonjour $nom_var  </h2>" ;}
     else echo "pas de cookie" ;
?>
```

Cookies & sessions

Suppression d'un cookie :

- Fonction `setcookie()` en spécifiant simplement l'argument NomCookie

```
<?php
setcookie("Visites");
?>
```

- Une autre méthode consiste à envoyer un cookie dont la date d'expiration est passée :

```
  <?php
setcookie("Visites","", )
?>
```

Cookies & sessions

Le tableau PHP **superglobal $_COOKIE** permet de **lire** tous les cookies transmis par le navigateur

Par défaut, sans date spécifiée, le cookie expire à la fin de la "session" (fermeture du navigateur)

Par **setcookie TIMESTAMP**, on peut spécifier une date à `setcookie`

Une date négative, signifie que le cookie ne disparaîtra jamais du client

Pour supprimer un cookie avec setcookie, on devra spécifier la valeur de `setcookie` à `null`

```
setcookie('id',null);
```

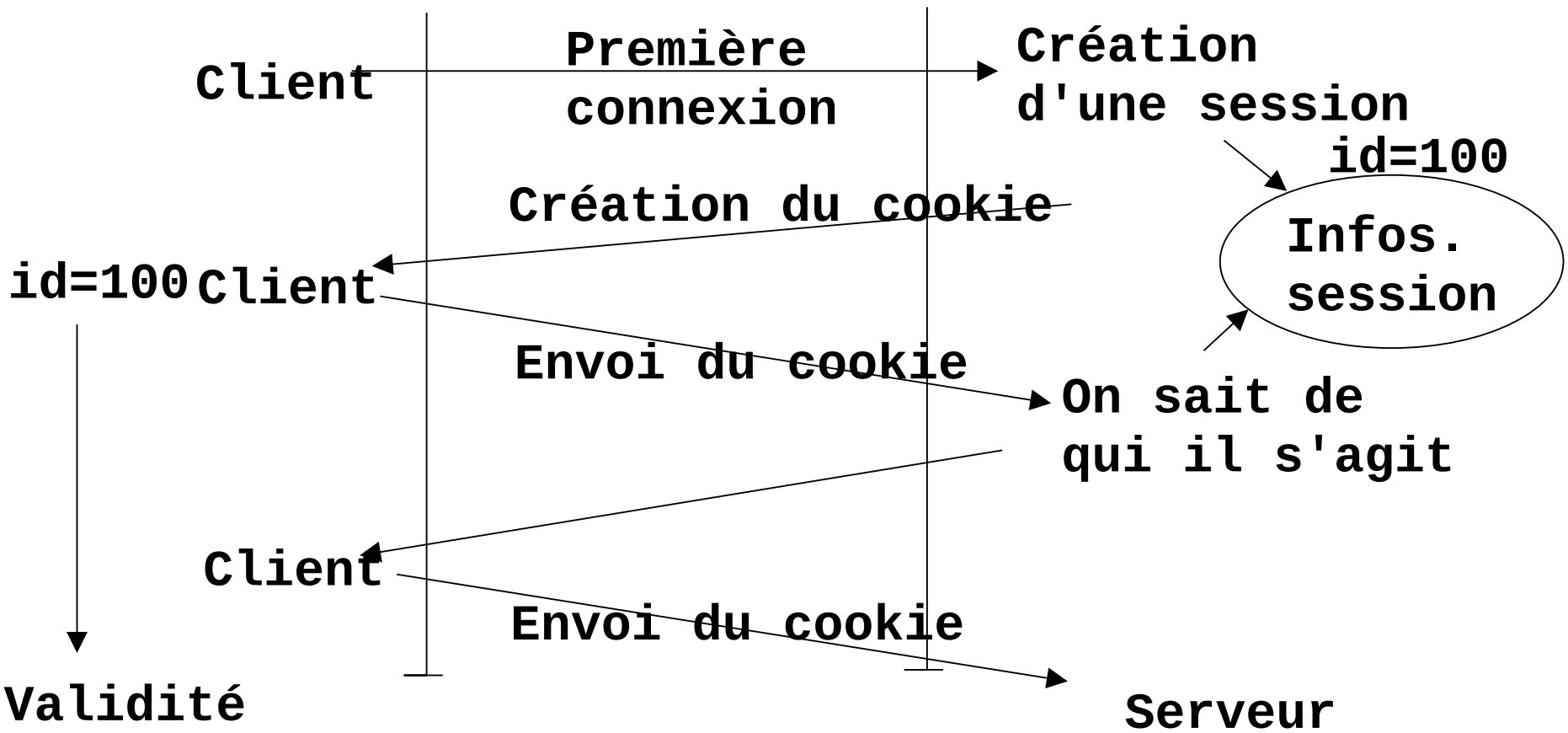
Par défaut, un cookie est visible par l'ensemble du site web qui l'a créé

Pour limiter un cookie à une page donnée, cette page doit être spécifiée dans le `setcookie`

```
//Le cookie ne sera visible que sur page.php
setcookie('id',5,null,'/page.php');
```

Cookies & sessions

Gestion de session avec cookies :



Cookies & sessions

Les informations des sessions sont stockées sur le serveur

Les informations des sessions sont donc moins sensibles que sur les cookies puisque “personne” ne peut accéder aux sessions sur le serveur

Pour démarrer une session, il faut, à chaque page, avant tout affichage, appeler la fonction :

```
session_start();
```

A chaque appel de `session_start`, php réalise une identification grâce à un identifiant de session (dans le dossier temp) :

si aucun cookie `SESSION_ID` n'a été crée, il crée un identifiant de session `sessid`

Cookies & sessions

Fonctions Php pour gérer les sessions :

`session_start()` :

- identifie ou crée la session
- si existe déjà, recrée toutes les variables Php associées
- doit être appelé au début du script Php

`session_destroy()`

- détruit les information associée à la session

Cookies & sessions

`session_id()` :

- renvoie l'identifiant de la session

`session_register(nomVar)` :

- associe une variable Php à la session

  -> cette variable sera retrouvée à chaque session

`session_unregister(nomVar)` :

- supprime une variable de la session

`session_is_registered(nomVar)` :

- test l'existence d'une variable

Cookies & sessions

Exemple 1: cookie1.php

```
<?php
    session_start();
    $toto=date("h:i:s");
    session_register(toto);
?>
```

Une visite de cette page -> mémorisation de $toto dans le navigateur

Cookies & sessions

Exemple 2 : cookie2.php

```
<?php
 session_start();
     if (session_is_registered(toto))
          echo 'ha ha ! Vous avez visite la
                    page cookie1.php le $toto';
     else
 echo 'premiere visite';
?>
```

Cookies & sessions

Une session peut être nommée :

```
session_name('nom');
```

Le cookie correspondant portera alors le nom renseigné

Une session peut être détruite, si et seulement si la session a été créée par un session_start :

```
session_destroy();
```

La session et le fichier de session sont détruits, mais pas le cookie qui existera tant que le navigateur n'est pas fermé

=> problème de sécurité

Il est recommandé de faire après :

```
setcookie(session_name(),null);
```

Cookies & sessions

La configuration se fait dans le fichier php.ini :

```
; Handler used to store/retrieve data.
session.save_handler = files
;session.save_path = "/tmp"
; Name of the session (used as cookie name).
session.name = PHPSESSID
; Initialize session on request startup.
session.auto_start = 0
; Lifetime in seconds of cookie or, if 0, until browser is restarted.
session.cookie_lifetime = 0
; The path for which the cookie is valid.
session.cookie_path = /
; The domain for which the cookie is valid.
session.cookie_domain =
; After this number of seconds, stored data will be seen as 'garbage' and
; cleaned up by the garbage collection process. (1440 seconds=24 minutes)
session.gc_maxlifetime = 1440
```

Pour modifier ces valeurs :

```
ini_set('propriété','valeur');
```

Pour récupérer une valeur :

```
ini_get('propriété');
```

Cookies & sessions

L'ensemble des informations de session est affecté à un tableau **$_SESSION**

On peut retrouver le fichier sur le serveur : sess_IDENTIFIANT+DE+SESSION.

Le contenu de $_SESSION étant sérializé (tranformation d'une variable en forme de chaîne), la fonction :

```
unserialize($_SESSION)
```

permet de transformer la chaîne de caractères pour en faire une variable