



# PROGRAMMATION PROCÉDURALE

(PHP/PYTHON)



# PARADIGMES DE PROGRAMMATION

---

Où l'on parle de programmation structurée, procédurale, orientée objet...

# PROGRAMMATION STRUCTURÉE

- Concept apparu dans les années 70 sous l'impulsion d'Edsger Dijkstra et popularisé par Nicklaus Wirth avec son langage Pascal, un des premiers langages 'sans GO TO'
- Le GO TO
  - Dans les langages à 'étiquettes'
  - A donné naissance à de sacrés plats de spaghettis !
- Introduit les boucles while(), for(), repeat until...
  - Un seul point de sortie par boucle..

```
DEBUT:
SI (A > B) GO TO SUITE
    ....
    ....
    A = A +1
    GO TO DEBUT
SUITE:
    ....
    ....
```

# PROGRAMMATION PROCÉDURALE

- Fondé sur l'appel de procédures et de fonctions,
- Décomposition du code en petits modules, si possible indépendants, pour améliorer sa maintenabilité et sa réutilisabilité.
  - « Small is beautiful », la devise d'UNIX écrit en langage C
- Les limites :
  - Notions de variables locales (un mieux) et globales 😞 ,
    - L'utilisation de variables globales compromet la maintenabilité et l'intégrité du code et des données.
    - Hum, les variables de session dans PHP.....
  - Encore beaucoup (trop) de code dupliqué dans des modules à fortes similarités

# PROGRAMMATION ORIENTÉE OBJETS (POO) (1/3)

- Objectif : Dépasser les limites de la programmation procédurale en terme de maintenabilité, d'intégrité et de réutilisabilité du code
- Au-delà de la rigueur induite par les langages à objets, certains logiciels, tels que les interfaces graphiques, ne pourraient être réalisés sans l'utilisation d'un langage OO.
- Concepts d'objets et de classes :
  - Un objet peut être abstrait ou être une représentation d'un 'objet' du monde réel (la taxe TVA à 20%, l'automobile de Mme Durand,...),
  - L'objet encapsule les données qui le caractérisent (attributs) ET son comportement (méthodes), c'est-à-dire sa manière d'interagir avec le monde extérieur,
  - Une classe modélise (définition des attributs et méthodes) les objets conceptuellement identiques (ex la classe 'automobile' dont l'automobile de Mme Durand représente une instance).

# PROGRAMMATION ORIENTÉE OBJETS (POO) (2/3)

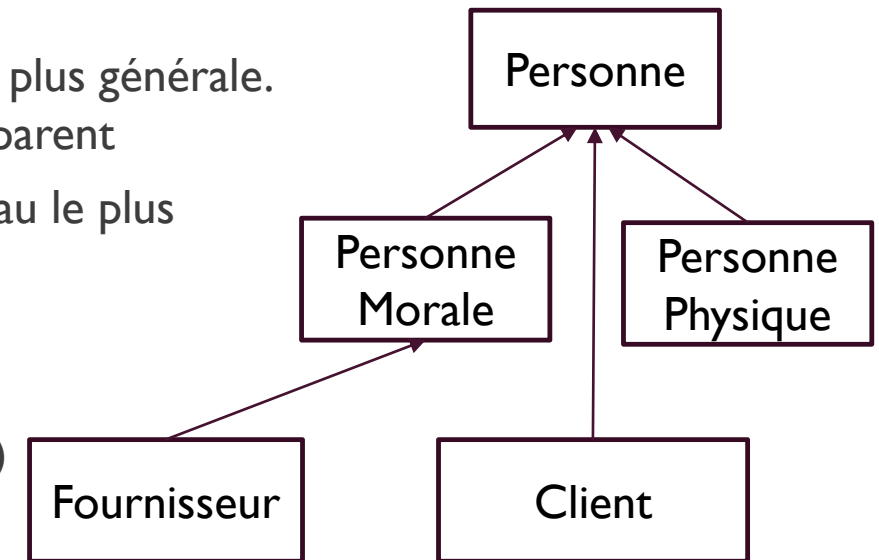
## ■ Ce qui différencie les langages à objets des autres (principes fondateurs)

- Encapsulation


- On ne peut lire et modifier les (certains) attributs d'un objet que via des méthodes propres à la classe de l'objet (getter/setter). Les données sont inaccessibles directement sauf si elles sont rendues publiques.

- Héritage

- Une classe peut être un sous-ensemble d'une classe de portée plus générale. La classe enfant 'hérite' des méthodes et attributs de la classe parent
- On définit les attributs et méthodes à un seul endroit : le niveau le plus générique possible :
  - ❖ Adresse au niveau Personne
  - ❖ SIRET/SIREN au niveau Personne Morale
  - ❖ CA réalisé au niveau Client (Personne Morale ou Physique)
  - ❖ Montant acheté au niveau Fournisseur (Personne Morale)



# PROGRAMMATION ORIENTÉE OBJETS (POO) (3/3)

- Ce qui différencie les langages à objets des autres (principes fondateurs)
  - Surcharge
    - Elle permet de définir plusieurs variantes d'une même méthode en fonction du nombre et de la nature des paramètres passés. Au sein de la même classe, on spécifie donc plusieurs méthodes portant le même nom et se différenciant par leurs arguments.  
 La notion d'argument facultatif c'est quand même bien pratique !
  - Polymorphisme
    - Permet de redéfinir au niveau d'une classe enfant une méthode déjà spécifiée dans la classe parent.

# EXEMPLES DE COUPLES PARADIGMES/LANGAGES

- Non structurée, non procédural
  - Assembleurs, FORTRAN, COBOL
- Procédurale
  - Pascal, Langage C, PL1, SQL (routines stockées), PHP
- Orientée Objet
  - C++, JAVA, JavaScript, Python, PHP,...



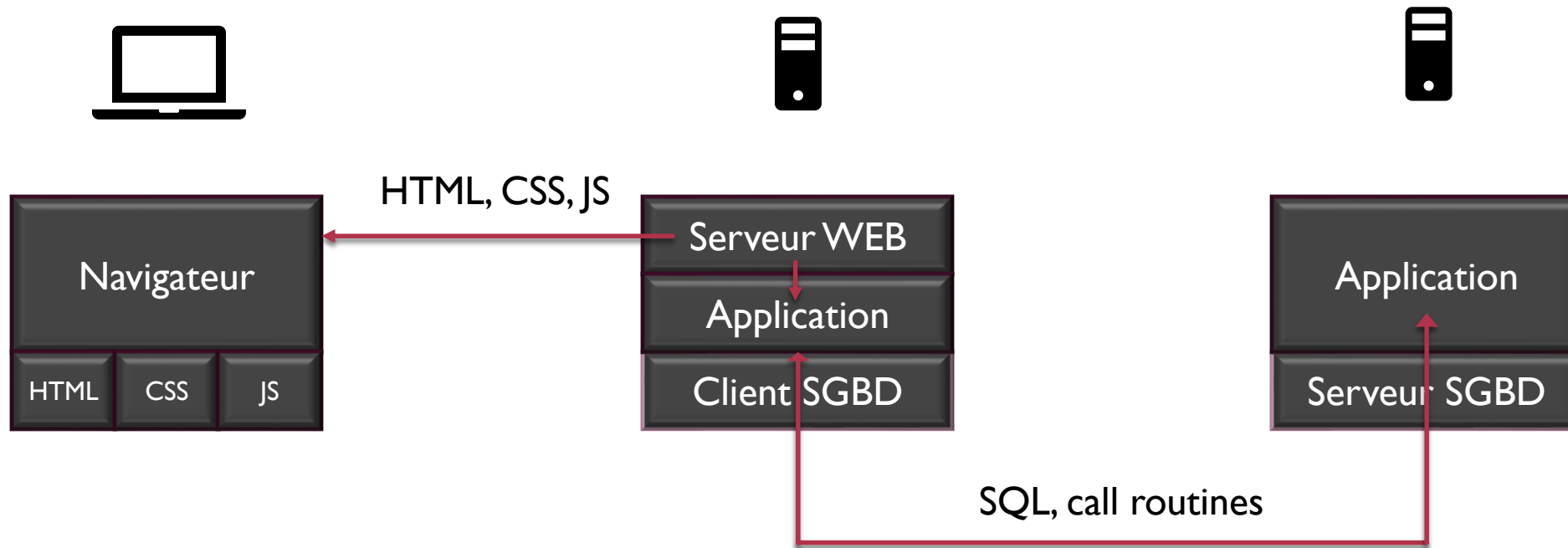


# ARCHITECTURES WEB

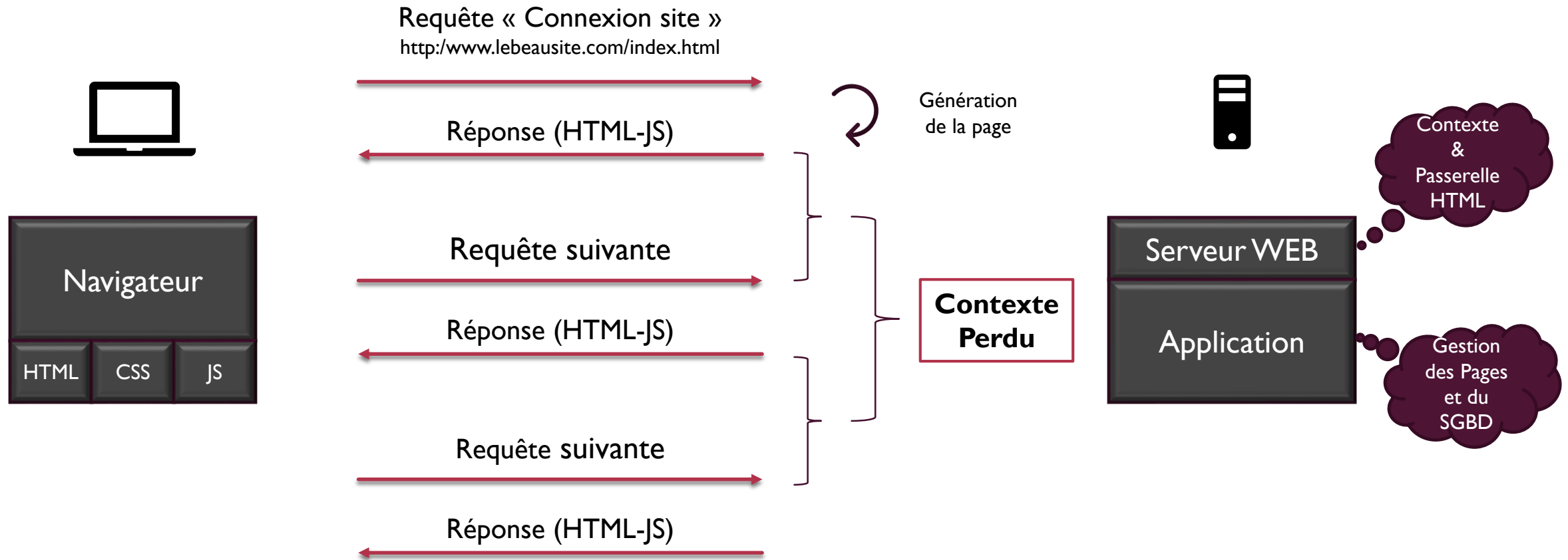
---

Quelques concepts

# ARCHITECTURES N TIERS



# ARCHITECTURES NON CONNECTÉES





# ENTERING PHP

---

Présentation, un peu d'histoire et installation environnement

# PRÉSENTATION PHP

## ■ PHP comme PHP Hypertext Processor

- Créé en 1995 par Rasmus Lerdorf, qui le baptise d'abord Personal Home Page Tools (bibliothèque de scripts lui permettant de savoir qui consulte son CV....)
- Un langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web.
- Permet de dépasser le côté statique du couple HTML/CSS
- Génère dynamiquement depuis le serveur du HTML, intégrant éventuellement des scripts JS, seuls capable d'être compris par le(s) navigateur(s).
- Fournit :
  - les fonctionnalités permettant d'interagir avec un SGBD,
  - et tellement d'autres choses....

# LES VERSIONS DE PHP (1/2)

## ■ Version 4 (Released 2000)

- Moteur PHP refondu par **Z**eev Suraski et **A**ndi Gutmans qu'ils baptise Zend Engine
- Intègre la gestion des sessions et amélioration des performances

## ■ Version 5 (Released 2004)

- Zend Engine II
- Support POO, dont la conséquence est l'apparition de PDO (PHP Data Objects) pour un accès universel aux SGBD
- Gestion des espaces de noms
- Extension Filter, pratique pour contrôler les saisies de formulaires

# LES VERSIONS DE PHP (I/2)

## ■ Version 7 (Released 2000)

- Zend Engine III : Amélioration très nette des performances du moteur
- Typage des paramètres passés et du paramètre retour des fonctions.
- ....

## ■ On trouve tout sur PHP :

- Le site de référence : <http://php.net>
- Le forum : <https://forum.phpfrance.com/>

# INSTALLER PHP

## ■ Il faudra d'abord un éditeur

- comme SublimeText : <https://www.sublimetext.com/>

## ■ Mettre en place l'univers PHP

- Serveur WEB, PHP, SGBD
  - Dans le monde de l'Open Source Apache (Server WEB) et MySQL sont les plus utilisés
  - Il faut les configurer pour qu'ils fonctionnent ensemble
    - ❖ Dans le monde Windows, WAMP (comme Windows, Apache, MySQL, PHP) est un package prêt à l'emploi très utilisé.  
On le télécharge ici : <http://www.wampserver.com/>





# PHP BEGINNING

---

Fonctionnement et structure de base

# PHP : PRINCIPE DE BASE

## ■ Encapsulation de balises PHP dans le HTML

```
<?php                                /* Balise ouvrante  (ceci est un commentaire  */  
    ..  
    Code PHP  
    ..  
?>                                /* Balise fermante                                */
```

- PHP n'interprète que ce qui est situé entre les balises

## PHP : PRINCIPE DE BASE (2/2)

- Le code PHP peut être intégré entre des balises <HTML></HTML> ou hors de ces balises
- Le code PHP peut être intégré entre des balises HTML de différentes natures
  - Ex :

```
<li>  
  <a class="menu" href="mailto:<?php echo($_SESSION['contactEmail']) ?>">  
    <?php echo($contact_label) ?>  
  </a>  
</li>
```

# PHP : 1<sup>ER</sup> EXEMPLE (1/2)

## ■ Code PHP

```
<!DOCTYPE html>
<html lang="fr" >
  <head>
    <title>Salutations</title>
    <meta charset="utf-8" />
  </head>
  <body>
    Hello world<br/>
    Nous sommes le :
    <?php
      setlocale(LC_TIME, 'fra_fra')      ;    /* Paramètres de dates en français */
      echo(strftime('%A %d %B %Y'))      ;    /* Format de la date souhaitée */
    ?>
  </body>
</html>
```

## PHP : 1<sup>ER</sup> EXEMPLE (2/2)

- **strftime()** : est une fonction qui formate sous la forme d'une chaîne de caractères la date/heure locale du serveur
- **echo()/echo** : renvoi du texte au navigateur
- **Notez :**
  - les commentaires :
    - `/*`  
commentaire sur plusieurs lignes possible  
`*/`
    - `//` commentaire sur une seule ligne
  - Le « ; » obligatoire en fin d'instruction
- **Au final : regarder le code générer sur la page affichée**
  - Click droit sur la page et quelque chose comme « Afficher le code source de la page »

# VISUALISER LES ERREURS PHP

- Le comportement par défaut de PHP est de n'afficher qu'une page blanche en cas d'erreur sur le script
  - Compréhensible en production
  - Gênant en développement !
- Configurer PHP pour l'affichage des erreurs, si le navigateur n'affiche pas les erreurs
  - Retrouver le fichier php.ini après exécution du script `<?php phpinfo() ;?>`
  - Le fichier se trouve sur la ligne '**Loaded Configuration File**'
  - Ouvrir le fichier et chercher '**error\_reporting**' et '**display\_errors**'
    - Dans php.ini un ';' un début de ligne est un commentaire
    - Ecrire **error\_reporting = E\_ALL** et **display\_errors = On**

# LES TYPES DE DONNÉES(I)

## ■ Type scalaire (une seule donnée à la fois)

- **string** ou chaîne de caractères : délimité entre quotes simples (') ou double(" )
  - Exemples :
    - ❖ 'Salut'
    - ❖ "la compagnie"
    - ❖ "C'est moi"
    - ❖ 'et ce n'est pas toi'
  - Les simples et doubles quotes ne sont pas tout à fait équivalentes, nous verrons la différence d'utilisation sur les opérations de concaténation de chaînes
- **int** ou entier
  - Exemples : -2, 23, 12387656.
  - Taille dépendante du serveur, en général limité à 32 bits,
  - Signé, donc valeurs comprises entre  $-2^{31}$  et  $+2^{31}$ 
    - ❖ Type 'BIGINT' des bases de données non utilisable

# LES TYPES DE DONNÉES(2)

## ■ Type scalaire (suite)

- **Réel signé** (nombres décimaux)

- Exemples : 2.45, -2.8, 1.4e3, 8E-10



- Le point décimal est le '.' et non la ','

- Flottant ou double, taille dépendante du serveur, en général limité à 64 bits (valeur maximale de 1.8e308)



- Attention aux erreurs de précision dus à l'utilisation de flottants :

- ❖  $2e121$  ne peut être différentié de  $(2e121 + 1)$

- **Booléen**

- **true** ou **false**



# LES TYPES DE DONNÉES (3)

## ■ Type composite (I)

### • tableau

#### ■ Exemples :

❖ Mono-dimension : `$couleurs = array('rouge','orange','jaune','vert','bleu','violet') ;`  
`$carte = array('nom' => 'DUPOND', 'prenom' => 'Jules') ; /* associatif */`



❖ Multi-dimension : `$annuaire = array(  
array('Pierre' , '06123457') ,  
array('Estelle' , '01234567') ,  
array('Max' , '98765432')  
) ;`

#### ■ Adressage

❖ par rang (le 1<sup>er</sup> élément à le rang 0 et non 1)

✓ `$couleurs[2] /* 'jaune' */`

❖ par clé

✓ `$carte['prenom'] /* 'Jules' */`

# LES TYPES DE DONNÉES (4)

## ■ Type composite (2)

- **objets**

- Exemples : `$myDB = new PDO()` /\* création d'un objet PDO pour la connexion à un SGBD \*/

## ■ Type ressource

Une ressource est une variable spéciale, contenant une référence vers une ressource externe. Les ressources sont créées et utilisées par des fonctions spéciales.

Typiquement descripteur de fichier retourné à l'ouverture d'un fichier.

- Exemple : `$MyFile = fopen('fichier_client.txt', 'r+') ;`

# LES VARIABLES (I)

- Un élément de base de tout langage de programmation
  - Sert à stocker (temporairement) une donnée
    - La donnée est perdue lorsque le programme s'arrête.
  - On peut également, **si cela a du sens**, lui appliquer des transformations (opérations)
    - Additionner une valeur à une variable numérique
    - Retirer les espaces non significatifs à une variable de chaîne

# LES VARIABLES (2)

## ■ Caractéristiques d'une variable (I)

- Son nom :
  - En PHP, un nom de variable, sensible à la casse, doit satisfaire les règles suivantes (cf. manuel PHP):  
« Les noms de variables suivent les mêmes règles de nommage que les autres entités PHP. Un nom de variable valide doit commencer par une lettre ou un souligné (\_), suivi de lettres, chiffres ou soulignés »
  - Exemple :
    - ✓ \$prenom
    - ✗ \$2main
  - Conventions de nom :
    - ❖ De manière à faciliter la lecture et la bonne compréhension des programme
    - ✓ On utilise des noms porteurs de valeur sémantique par rapport au contenu de la variable,
    - ✓ Pour nommer une variable qui contient plusieurs mots il existe des recommandations :
      - Le Camel case : \$nomClient
      - Le Pascal case : \$NomClient
      - L'underscore case : \$nom\_client

# LES VARIABLES (3)

## ■ Caractéristiques d'une variable (2)

- Son type :
  - PHP est faiblement typé
    - ❖ Pas de déclaration de variable nécessaire, cela est très pratique pour certains, le comble du manque de rigueur pour d'autres.
    - ❖ Le typage se fait automatiquement en fonction du type de donnée contenue donné à l'affectation
      - ✓ `$nom = 'toto' /* on se doute que c'est une chaine de caractères */ ;`
      - ✓ `$i = 2 /* un entier, mais peut être qu'en réalité, relativement à la sémantique de cette variable, elle pourrait contenir ultérieurement un décimal ! */ ;`
      - ✓ `$tel = null /* null est un marqueur spécial qui précise que la variable ne contient rien (et donc n'est pas typé) */ ;`

remarque: `$string = "" /* chaine vide */`  
et `$string = null` n'ont rien d'identique.

# LES CONSTANTES

## ■ Stop aux valeurs codées en dur !

- Plus jamais  $\$MontantTVA = 0.2 * \$MontantHT$  ;

- Mais :

```
?php
```

```
define('TAUX_TVA',0.20) ;
```

```
...
```

```
 $\$MontantTVA = TAUX\_TVA * \$MontantHT;$ 
```

```
..
```

```
?>
```

- Par convention on écrit les constantes en majuscules

# LES OPÉRATEURS (I)

## ■ Affectation

- On valorise (« affectation ») une variable avec l'opérateur '='

<?php

\$montant = 2000 /\* la variable \$montant est un entier qui vaut maintenant 2000 \*/ ;

\$oldTva = 19.6 /\* la variable \$oldTva est un decimal qui vaut maintenant 19.6 \*/ ;

\$libelle = 'montant' /\* la variable \$libelle est un string qui vaut maintenant 'montant' \*/ ;

?>

# LES OPÉRATEURS (2)

## ■ Opérateurs de base ('+', '-', '\*', '/', '%')

```
<?php
    $a  = 2          ;
    $b  = $a + 4     ;
    echo $b          ;    /* $b vaut 6 */
    $a  = $b / $a     ;
    echo $a          ;    /* $a vaut 3 */
    $c  = $a % 2      ;    /* Reste de la division de $a par 2 */
    echo $c          ;    /* $c vaut 1 */
    $c  = $a * $b     ;
    echo $c          ;    /* $c vaut 18 */
?>
```



# LES OPÉRATEURS (3)

## ■ Incrémentation et Décrémentation

```
<?php
    $a      = 1      ;
    ++$a    ;        /* pré incrémentation */
    echo    $a      ;        /* $a vaut 2 */
    $b      =  $a++   ;        /* post incrémentation de $a */
    echo    $b      ;        /* $b vaut 2 */
    echo    $a      ;        /* $a vaut 3 */
    echo    $a--     ;        /* echo affiche 3 */
    echo    --$a     ;        /* echo affiche 1 */
?>
```

# LES OPÉRATEURS (4)

## ■ La concaténation

- On utilise le ‘.’

```
<?php
```

```
$politesse      = 'Bonjour'      ;
```

```
$civilite       = 'Monsieur'     ;
```

```
$formule        = $politesse.' '.$civilite ;
```

```
$formule        .= ' Durand'      ;    /* combinaison affectation et concatenation */
```

```
echo            $formule          ;    /* ‘Bonjour Monsieur Durand’ */
```

```
?>
```

# LA COMPARAISON (I)

## ■ Opérateurs de base

- **Égalité** : '==' ex: \$a == \$b /\* ne pas confondre avec l'opérateur d'affectation '=' \*/

Ce test renvoie un booléen true si après transtypage \$a vaut \$b et false dans le cas contraire

**Inégalité** : '!=' ex: \$a != \$b

Ce test renvoie un booléen false si \$a vaut \$b et true dans le cas contraire

**Identité** : '===' ex: \$a === \$b

Ce test renvoie un booléen true si \$a vaut \$b et que \$a et \$b sont de même type, false dans le cas contraire ('!=' existe également)

**Infériorité** : '<' ou '<=' si on teste également l'égalité ex: \$a < \$b

Ce test renvoie un booléen true si \$a est strictement inférieur à \$b et false dans le cas contraire

**Supériorité** : '>' ou '>=' si on teste également l'égalité ex: \$a > après \$b

# LA COMPARAISON (2)

## ■ Composition des opérations de comparaison

- Le test qui compose des comparaison à base d'opérateurs ET (AND ou &&) et OU (OR ou ||) va renvoyer true ou false
- Utiliser les parenthèses pour être clair sur les priorités de test

( (A ||B) && (C||D) )

Évalue: d'abord si A ou B qui donne un booléen B1  
          puis si C ou D qui donne un booléen B2  
          puis si B1 et B2

# LA COMPARAISON (3)

## ■ La structure de base : if....else (I)

- Effectue le test de base, sa syntaxe est la suivante :

```
if (condition)
{
    /* le code à exécuter si la condition est vraie */
}
else {
    /* le code à exécuter si la condition est fausse */
}
```

# LA COMPARAISON (3)

## ■ La structure de base : if....else (2)

- Exemple tester et afficher si un nombre est pair ou impair

```
<?php
```

```
$nombre = 4 ;
```

```
if **A COMPLETER**
```

```
...
```

```
...
```

```
?>
```

# LA COMPARAISON (4)

## ■ La structure de base étendue : if....elseif..else (I)

- Syntaxe :

```
if (condition)
```

```
{  
    /* le code à exécuter si la condition est vraie          */  
}
```

```
elseif (nouvelleCondition)
```

```
{  
    /* le code à exécuter si 'condition' est fausse ET 'nouvelleCondition' est vraie */  
}
```

```
else
```

```
{  
    /* le code à exécuter si 'condition' est fausse ET 'nouvelleCondition' est fausse */  
}
```

# LA COMPARAISON (5)

## ■ La structure de base étendue : if....elseif...else (2)

- Exemple tester et afficher si un nombre est pair, premier, ou impair

```
<?php
function EstPremier($argument)
{
    $result = false ;
    for($i = 2; ($i < $argument) && ($argument%$i != 0); $i++)
    {}
    if ($i == $argument)
    {
        $result = true ;
    }
    return $result ;
}
$
nombre = 17 ;
** A Compléter **
```



# LA COMPARAISON (6)

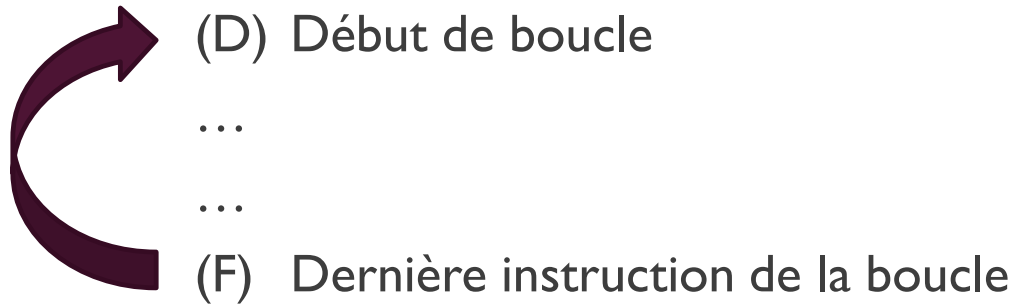
## ■ switch()

- Pratique pour éviter les cascades de if..elseif..elseif...elseif

```
switch($argument)
{
    case 1:
        /* code à exécuter si $argument prend la valeur 1 */
        break;      /* pour éviter de tester la valeur à case 2, case ... */
    case 2:
        /* code à exécuter si $argument prend la valeur 1 */
        break
    ...
    default:
        /* code à exécuter par défaut */
}
```

# LES BOUCLES (I)

## ■ Principe



- En général on définit une condition de sortie de la boucle

# LES BOUCLES (2)

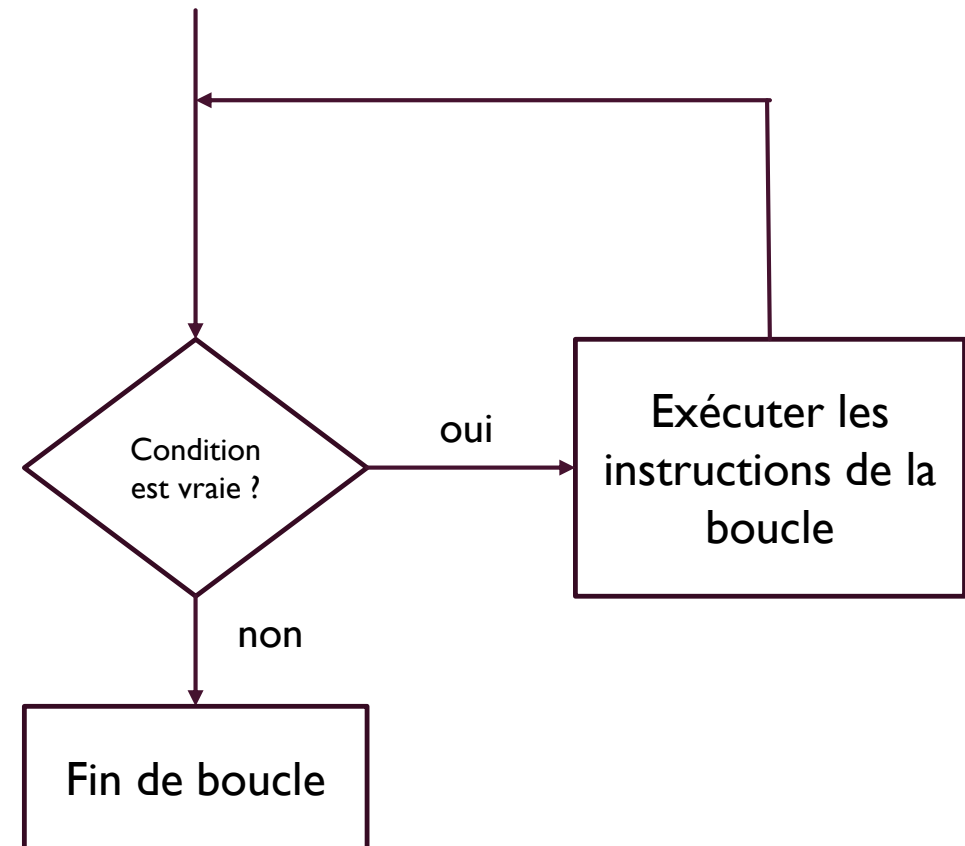
## ■ while()

**Syntaxe :** while(ConditionIsTrue)

```
{  
    /* code à exécuter */  
}
```

Exemple :

```
<?php  
$repeter = 10 ;  
while($repeter > 0)  
{  
    echo 'Vive le PHP !<br/>' ;  
    $repeter-- ;  
}  
?>
```



# LES BOUCLES (3)

## ■ do-while()

**Syntaxe :**

```
do
{
/* code à exécuter */

} while(ConditionIsTrue)
```

identique à while() sauf que la boucle est exécutée au moins une fois.

# LES BOUCLES (4)

- **for()** : idem while(), intègre en plus un compteur

**Syntaxe** : **for** (initialisationCompteur; condition; incrémentationCompteur)

**Exemple** : `for ($i=0; $i < MAX; $i++)`  
    {  
        /\* code boucle \*/  
    }

Refaire l'exemple while() précédent avec une boucle for()

# LES BOUCLES (5)

- **foreach()** : permet de travailler sur chacun des éléments d'un tableau sans connaître son nombre d'éléments.

**Syntaxes** :   **foreach** (tableau as \$Value) /\* tableau simple \*/  
              **foreach** (tableau\_associatif as key => \$Value) /\* tableau associatif \*/

```
Tester avec : <?php  
$Fiche = array('Nom' => 'Dupond', 'Prenom'=>'Marcel') ;  
puis avec $Annuaire = array (  
    array('Nom' => 'Dupond', 'Prenom'=>'Marcel') ,  
    array('Nom' => 'Dumoulin', 'Prenom'=>'Emile')  
);
```

# LES FONCTIONS (I)

- Les fonctions permettent de centraliser un traitement spécifique, comme par exemple « obtenir les données d'un client ». Ceci favorise :
  - La réutilisabilité du code,
  - La maintenabilité évolutive et correctrice,
  - L'écriture de scripts plus clairs et concis.
- Il existe 2 types de fonctions
  - Les fonctions natives (ex : **strftime()**, **phpinfo()**,... ),
  - Les fonctions utilisateurs développées en propre pour les besoins d'une application

)

;

# LES FONCTIONS (2)

## ■ Déclaration d'une fonction utilisateur :

Syntaxe : `function nomFonction(Argument1,Argument2,...ArgumentN)`

```
{  
    /* Code fonction      */  
  
    ...  
    return($Valeur) ;  
}
```

Notes : - Une fonction retourne une valeur  
- Depuis PHP 7.0, on peut forcer le contrôle de type au niveau arguments et valeur retournée

Exercice : Ecrire une fonction qui calcule le volume d'un parallélépipède rectangle ( $a*b*c$ )  
;



# LES FONCTIONS (3)

## ■ Les fonctions prêtes à l'emploi

C'est un des points fort de PHP qui propose des centaines de fonctions prêtes à l'emploi :

- Fonctions de date et heure,
- Gestions de chaines de caractères,
- Envoi de fichiers ou de mail,
- Gestion de mots de passe,
- Mathématiques,
- Traitement d'images,....

SE REPORTER A <http://www.php.net/manual/fr/>



# PHP – TELLEMENT ESSENTIEL

---

Portée des variables – Inclusion de code – Coder en professionnel

# PORTÉE DES VARIABLES (I)

- La portée d'une variable (ie où est-elle visible ?) dépend du contexte dans lequel la variable est définie
- La portée peut être :
  - Locale,
  - Globale,
  - Superglobale.

# PORTÉE DES VARIABLES (2)

## ■ Locale - Principe

- Pour la majorité des variables, la portée concerne la totalité d'un script PHP. Mais, lorsque vous définissez une fonction, la portée d'une variable définie dans cette fonction est locale à la fonction. Inversement, une variable locale définie dans un script en dehors d'une fonction de ce script n'est pas visible à l'intérieur de cette même fonction.

```
<?php
    $a = 1          ;    /* $a est locale au script          */
    $c = $a + $b    ;    /* génère une erreur 'Undefined variable' */

    function test()
    {
        $b = 2      ;    /* $b est locale à la fonction          */
        echo $a + $b ;    /* génère une erreur Undefined variable */
    }
    test();
?>
```

# PORTÉE DES VARIABLES (3)

## ■ Locale - Durée de vie

- Par défaut, une variable locale est instanciée à chaque exécution du script ou de la fonction. Dans le cas d'une fonction, si elle est déclarée comme 'static', la donnée contenue dans la variable est persistante.

```
<?php
function test()
{
    static $a  = 0      ;
    echo $a    ;
    $a        = $a  + 2  ;
}
test() ;
test() ;
?>
```

# PORTÉE DES VARIABLES (4)

## ■ **superglobal**

- Les Superglobales sont des variables internes qui sont toujours disponibles, quel que soit le contexte
- La liste des variables superglobales est prédéfinie et finie. Nous en verrons l'usage ultérieurement
  - \$GLOBALS
  - \$ \_SERVER
  - \$ \_GET
  - \$ \_POST
  - \$ \_FILES
  - \$ \_COOKIE
  - \$ \_SESSION
  - \$ \_REQUEST
  - \$ \_ENV

# PORTÉE DES VARIABLES (5)

## ■ Global (I)

- La portée d'une variable **globale** permet, à l'intérieur d'une fonction, de référencer une variable définie dans le script à l'extérieur de la fonction.

```
<?php
$a = 1;
$b = 2;
function somme()
{
    global $a, $b ;
    $b = $a + $b ;
}
somme() ;
echo $b ;    /*    $b vaut 3 */
?>
```

# PORTÉE DES VARIABLES (6)

## ■ Global (2)

- La variable superglobal `$GLOBALS`, qui est un tableau associatif dont les clés sont les noms des variables globales et les valeurs, les valeurs des variables globales peut également être utilisée.

```
<?php
$a = 3;
$b = 2;
function somme()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}
somme()      ;
echo $b      ;           /* $b vaut 5 */
?>
```



# INCLUSION DE FICHIERS (I)

- Inclure des fichiers permet de mutualiser son code et de l'organiser sous forme de bibliothèques.
  - Constantes générales,
  - Accès base de données,
  - Fonctionnelles
    - Accès utilisateurs,
    - Règles fonctionnelles...
    - ...
  - Portions de pages HTML...
  - ...

# INCLUSION DE FICHIERS (2)

- 4 instructions pour inclure :
  - `include()`, `require()`, `include_once()`, `require_once()`
- Les fonctions `*_once` n'incluent qu'une et une seule fois les fichiers cible dans le script.
- `include*` et `require*` traitent différemment l'absence de fichier :
  - `include` retourne un warning
  - `require` retourne une erreur fatale (arrêt du script)

# INCLUSION DE FICHIERS (3)

- Inclusion de portions de page HTML (I)
  - L'inclusion de portions de pages incluses dans des fichiers php permet d'éviter de coder le même code HTML pour toutes les pages notamment pour ce qui concerne :
    - Header,
    - Menus,
    - Footer
    - Autres sections banalisées sur le site.
  - Un script PHP (fichier .php) peut très bien contenir exclusivement du code HTML

# INCLUSION DE FICHIERS (3)

```
(<!DOCTYPE html>
<html>
  <head>
    ...
  </head>
  <body>
    <?php require("entete.php"); ?>
    <?php require("menus.php"); ?>
    <div id="the page">
      <h1>Ma Page</h1>
      <p>
        Bienvenue ...
      </p>
    </div>
    <?php require("pied_de_page.php"); ?>
  </body>
</html>
```

# CODER EN PROFESSIONNEL...

- Liste non exhaustive de (fortes) recommandations
  - Penser que le code sera relu par une autre personne que l'auteur originel.
    - Commenter le code,
    - Utiliser des noms de variables et fonctions qui facilitent la compréhension,
      - ❖ Utiliser un préfixage permettant de comprendre tout de suite à quoi la variable ou la fonction se réfère par exemple `clients_getContactInformations()`
    - Indenter le code,
    - Gérer les erreurs,
    - 'Never trust users !'
      - ❖ Faille XSS (Cross Site Scripting)s
      - ❖ Inclusion SQL
      - ❖ ...
    - Recommandations 'officielles' : <http://www.php-fig.org/psr/psr-1/> et <http://www.php-fig.org/psr/psr-2/>

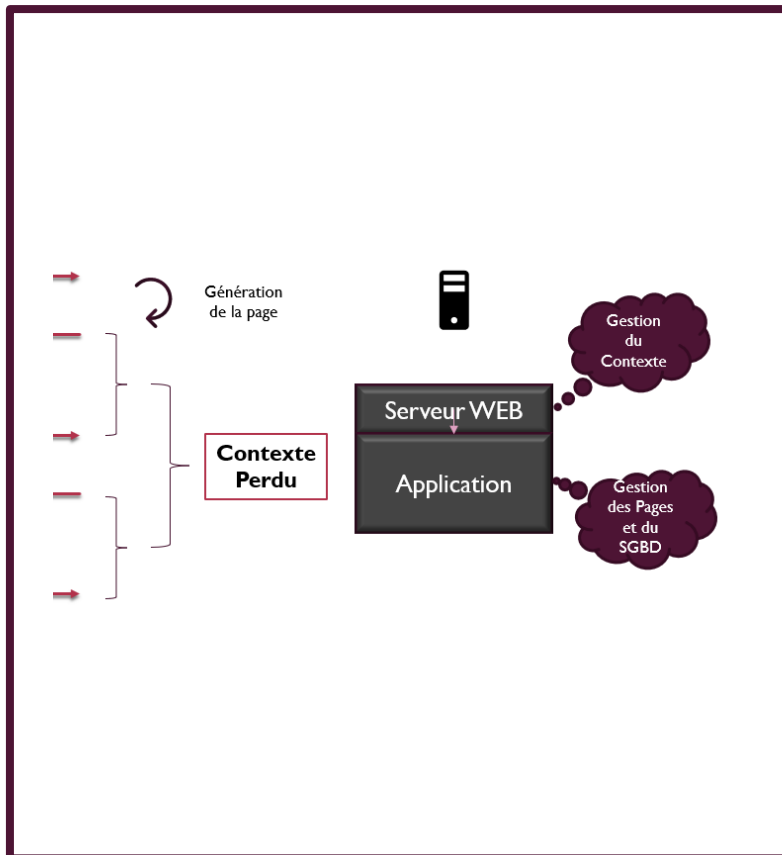


# PHP - ÉCHANGES INTER-SCRIPTS

---

Mécanismes de base

# TRANSMISSION DE DONNÉES INTER-PAGES (I)



## ■ Positionnement du problème

- A l'origine http a été conçu pour la consultation de documents. C'est un protocole d'hypertexte, chaque page contient la référence (URL) d'autres pages consultables sur le WEB.
- Chaque page lu était indépendante de la page lu précédente et de la page à venir.
- Dans les utilisations plus évoluées du Web il est devenu nécessaire de conserver un contexte d'échange tel que l'identifiant de l'utilisateur, ses droits d'accès, voire bien d'autres informations inavouables...

# TRANSMISSION DE DONNÉES INTER-PAGES (2)

- 4 méthodes pour transmettre des données entre différentes pages
  - Via l'URL,
  - Via l'utilisation d'un formulaire,
  - Via des variables superglobales (variables de Session en particulier),
  - Via les cookies



# TRANSMISSION DE DONNÉES INTER-PAGES (3)

## ■ Transmission par l'URL (Uniform Resource Locator)

- Syntaxe : `<url_page>?nom_parametre1=valeur_parametre1&.....&nom_parametreN=valeur_parametreN`
  - ❖ Le '?' sépare l'url de la page des paramètres à suivre
  - ❖ Le '&' sépare les paramètres (Attention dans les URL référencées dans les pages le '&' doit être échappé par &amp;)
  - ❖ La longueur de l'URL doit être limitée à 255 caractères (!)
- Récupérer les paramètres passé en PHP
  - ❖ PHP utilise la superglobal `$_GET` : les paramètres sont stockés dans cet array.
  - ❖ `$_GET['nom_parametre']` fourni la valeur\_paramètre
  - ❖

## ■ Exercice

- Afficher les nom et prénom passés en paramètre à une url :
  - `Votre_nom?nom=LE GRAND&prenom=Alexandre`

# TRANSMISSION DE DONNÉES INTER-PAGES (3)

## ■ Tester la présence d'un paramètre

- On utilise la fonction standard isset()

Syntaxe :     bool isset(mixed \$var [, mixed \$...] )

❖ Définit si une variable est définie et non NULL

## ■ Tester le type du paramètre

- On utilisera les fonctions standards is\_string(), is\_int(), is\_float(), filter\_var()...

## ■ Exercice

- Refaire l'exercice précédent en testant les variables saisies

# TRANSMISSION DE DONNÉES INTER-PAGES (4)

- Passer des paramètres par l'url est-il prudent ?
  - Faille XSS
  - ...

# TRANSMISSION DE DONNÉES INTER-PAGES (5)

## ■ Transmission par formulaire

- Rappel : dans un formulaire on précise la page qui traitera les données saisies après validation

```
<form method="post" action="handler_post.php">
```

/\* la methode « post » est préférée  
à la méthode « get » où l'on ne peut récupérer  
plus de 255 caractères \*/

```
....
```

```
....
```

```
</form>la valeur_paramètre
```

- La page 'handler\_post.php' se chargera de récupérer les données via la **superglobale \$\_POST**

# TRANSMISSION DE DONNÉES INTER-PAGES (6)

- Valider les données entrées avec `filter_var()`
  - Fonction très puissante et complète, se reporter à <http://php.net/manual/fr/function.filter-var.php>
- Exercices
  - Exercice 1
    - Afficher les nom et prénom passés via un formulaire
    - Utiliser la fonction **`htmlspecialchars()`** pour se protéger de XSS.
  - Exercice 2
    - Ajouter le champ email et contrôler que le champ saisi est bien un email avec `filter_var()`

# UPLOAD DE FICHIERS (I)

- L'envoi se fait via l'utilisation d'un formulaire qui comprend :
  - Dans le tag <form> l'attribut enctype="multipart/form-data",
  - Un champ <input type="file" name="the\_file"/>
- Dans le script 'post' on récupère les caractéristiques du fichier par la superglobale \$\_FILES
  - Une variable \$FILES['the\_file'] est créée,
  - \$\_FILES['the\_file']['name'] contient le nom du fichier (full pathname)
  - \$\_FILES['the\_file']['type'] contient le type mime du fichier (cf. [https://fr.wikipedia.org/wiki/Type\\_de\\_m%C3%A9dias](https://fr.wikipedia.org/wiki/Type_de_m%C3%A9dias) )
  - \$\_FILES['the\_file']['size'] contient la taille en octets du fichier (en PHP taille max = 8 Mo)
  - \$\_FILES['the\_file']['tmp\_name'] repertoire temporaire sur lequel le fichier est placé avant acceptation,
  - \$\_FILES['the\_file']['error'] code erreur en cas de problème (0 = OK),

## UPLOAD DE FICHIERS (2)

- L'acceptation du fichier se fait via la fonction `move_uploaded_file()`

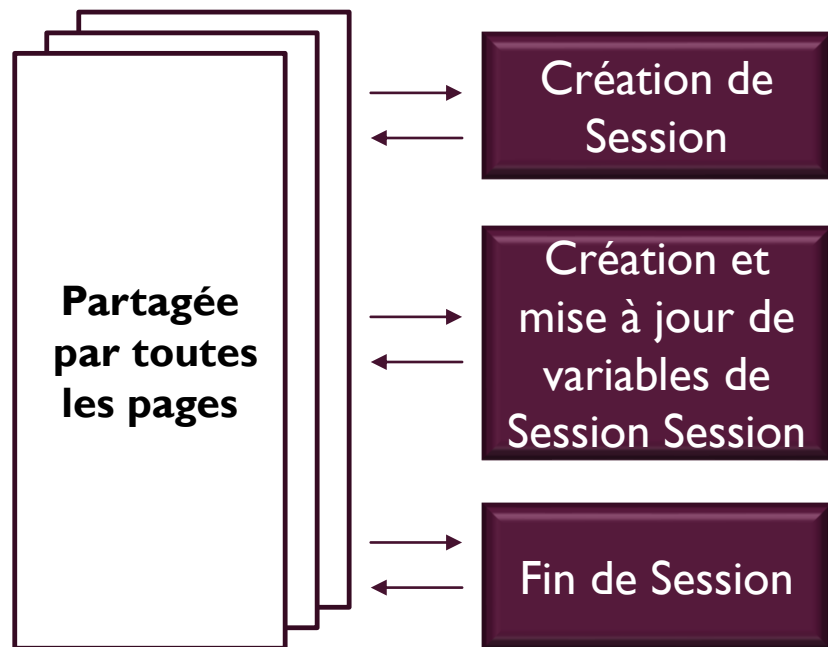
Syntaxe : `bool move_uploaded_files (string $filename, string $destination)`

Remarque : comme `$_FILES['the_file']['name']` contient le full pathname du fichier qui a été téléchargé, il faut, en général, extraire son nom via la fonction **`basename($_FILES['the_file']['name'])`** pour construire `$destination`.

- Exercice :
  - Charger dans le repertoire 'uploaded\_files' un fichier dont l'extension fait partie de la liste suivante (pdf, jpg, jpeg, gif, png, php). On utilisera les fonctions 'in\_array()' et `pathinfo()`.

# VARIABLES DE SESSION (I)

- La session permet de conserver des données, et fait un contexte entier, d'une page à l'autre. Cela permet de dépasser le cadre de GET et POST qui sont plutôt conçues pour transmettre de l'information parcellaire.



<code>session_start()</code>	Numéro attribué à l'arrivée sur le site et partagé par les pages visitées, l'appel doit être fait en tout début de page avant tout code HTML
<code>\$_SESSION</code>	Superglobale (array) où sont enregistrées les variables de session. Création et mise à jour à volonté
<code>session_destroy()</code>	ferme la session. Appelée implicitement lorsque l'internaute reste inactif plusieurs minutes



# VARIABLES DE SESSION (2)

## ■ Avantages :

- Permet de passer des données entre pages avec plus de discrétion que via l'URL,
- Pas de limites sur le nombre de variables
- Gestion d'un contexte global possible (ex : panier en cours sur un site marchand)

## ■ Limites :

- Pas utilisable pour les objets,
- L'utilisation de variables globales est un frein à la lisibilité, la maintenabilité, la réutilisabilité du code.


# VARIABLES DE SESSION (3)

- Exercice : reprendre le formulaire Nom, Prénom et passer les données par les variables de session
  - On utilisera un troisième script pour faire l'affichage
    - Activation par : `header('Location: script.php')`

# COOKIES (I)

- Par opposition aux variables de session qui gère les données, sans persistance, sur le serveur un cookie est un fichier enregistré, donc persistant, sur le poste de travail de l'internaute par le navigateur.
- En général on ne stocke qu'une seule donnée sur un cookie.
- Les cookies ont :
  - Un nom,
  - Une valeur,
  - Une date d'expiration  
Elle est en fait un timestamp qui mesure le nombre de secondes écoulées depuis le 01/01/1970. La valeur du timestamp en cours (Maintenant – 01/01/1970) vaut `time()`. Pour définir la date d'expiration d'un cookie il faut donc ajouter à `time()` le nombre de secondes avant expiration. La valeur d'un cookie qui expire dans une semaine vaut donc : `time() + 7*24*3600`

# COOKIES (2)

- Les cookies viennent du poste de travail 
  - A ce niveau n'importe quel utilisateur peut créer un faux cookie sur son poste de travail alors méfiance.....

# COOKIES (3)

## ■ Création d'un cookie

Syntaxe : `setcookie(NomCookie,ValeurCookie,TimestampExpiration)`

Exemple : `setcookie('Client','Jo le Taxi', time()+7*24*3600) ;`

Cette fonction possède également d'autres arguments facultatifs dont 2 sont importants en matière de sécurité

- `[Secure bool $secure=false]`  
Positionné à true indique que le cookie ne doit être envoyé que par une connexion sécurisée https. C'est au développeur de s'en assurer (!) (il peut utiliser la superglobale `$_SERVER('HTTPS')` pour vérifier)
- `[httponly]`  
Positionné à true, le cookie ne sera accessible que par le protocole HTTP (pas via du script comme JavaScript par ex). Protection contre XSS ! Attention non supporté par tous les navigateurs !

# COOKIES (4)

## ■ Suppression d'un cookie

Syntaxe : `unset($_COOKIE['nom_du-cookie']) ;`

Si, si cela existe !!! Il est possible de nettoyer derrière soi....

## ■ Exercice : créé un cookie, nom et valeur au choix...

- On vérifiera sa présence dans le navigateur.
- On retrouvera le cookie dans un autre script.

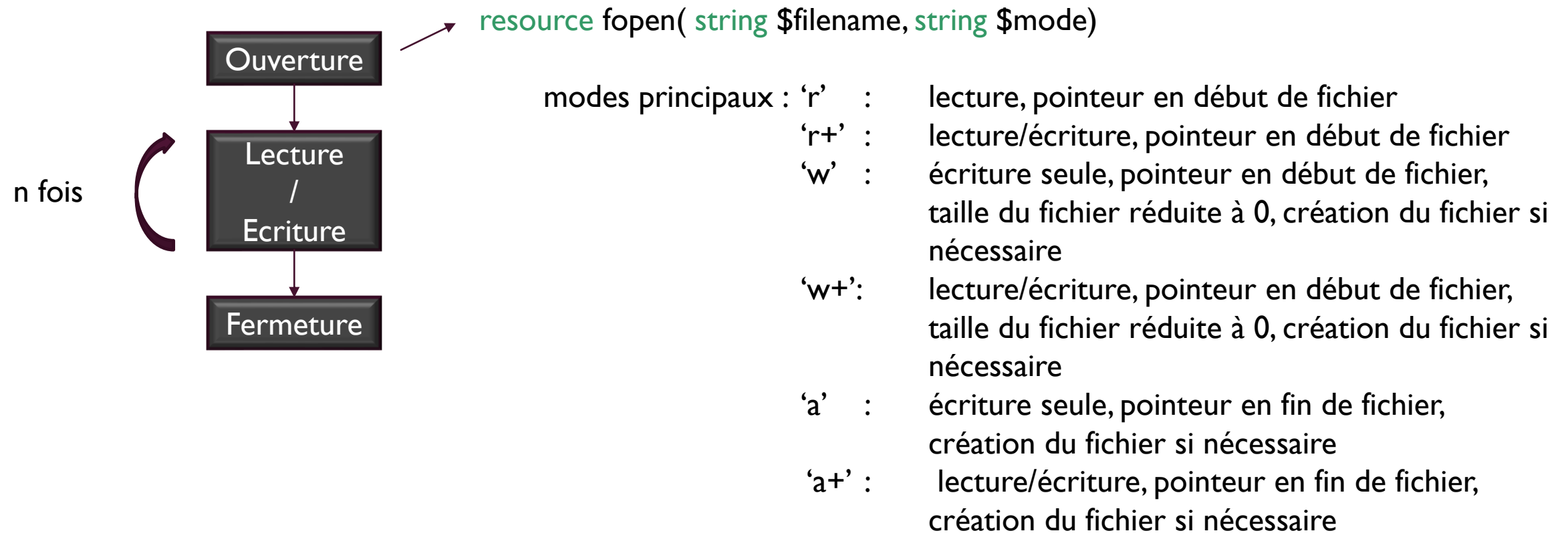
# LES FICHIERS (I)



- **Droits accès aux fichiers sur Linux**
  - Sous Linux les droits d'accès d'un utilisateur à un dossier ou un fichier sont structurés en droits pour son propriétaire, son groupe, les autres.
  - On définit pour chacun de ces droits d'accès les droits :
    - Pour un fichier, en lecture, écriture et exécution
    - Pour un dossier en lecture ou écriture
- **Le serveur PHP doit avoir le droit d'écrire dans le dossier serveur et de modifier les fichiers**

# FICHIERS (3)

## ■ Cycle de travail sur un fichier





# FICHIERS (3)

## ■ Lecture de fichier

- `fgetc()` lecture caractère à caractère.
- `fgets()` lecture ligne à ligne. Une ligne est une suite de caractère qui se termine pas un saut de ligne.
- `fread()` lecture en binaire sur une longueur.

## ■ Ecriture de fichier

- `fwrite()` écriture en binaire d'un string limitée facultativement à une longueur.

## ■ Fermeture du fichier

- `fclose()`

# FICHIERS (4)

## ■ Quelques fonctions utiles

- `pathinfo()`      Retourne des informations sur un Path fichier (Nom, Dossier, Extensions,...).
- `filesize()`      Renvoi la taille du fichier en octets
- `file_exists()`    Vérifie si un fichier ou dossier existe
- `file_type()`      Renvoi le type de fichier

## ■ Exercice

Reprendre l'exercice 'Upload File' et, une fois le fichier chargé, en faire une copie dans le dossier 'files\_copies'.



# BASES DE DONNÉES RELATIONNELLES

---

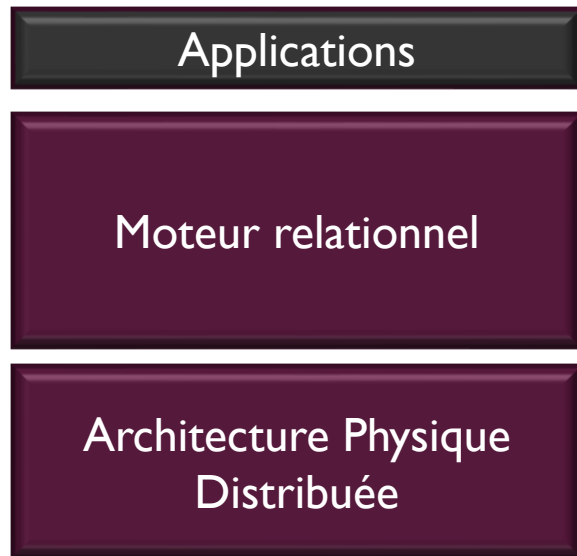
## Introduction

# ELÉMENTS DE DÉFINITION

## ■ Une base de données relationnelle :

- Est une collection de données organisées formellement sous la forme de relations, le terme relation devant être entendu au sens mathématique de la théorie des relations elle-même construite à partir de la théorie des ensembles.
- S'appuie sur un moteur relationnel, capable de manipuler des relations, c'est-à-dire des ensembles de données. En d'autres termes, dans le monde du relationnel on ne manipule pas des éléments mais des ensembles ! Le moteur relationnel traite les questions qu'on lui formalise sous la forme d'opérations d'algèbre relationnelle. Plus simplement, un moteur relationnel sait répondre à la question suivante : « Parmi l'ensemble des relations composant ma base de données, trouve moi l'ensemble (relation) qui a les caractéristiques suivantes ».
- En corolaire, en relationnel on ne manipule pas des 'enregistrements' mais des ensembles entiers.
- Modèle relationnel inventé par Edgar Frank « Ted » Codd en 1970 et construit et théorisé jusqu'au-delà des années 2000 (6NF en 2003)....

# ELÉMENTS D'ARCHITECTURE



- Réalise l'indépendance entre le modèle logique (relationnel) et l'implantation physique (serveur, grappes, disques, fichier, organisation logique à l'intérieur du fichier, découpage physique du fichier, gestion index,...),
- Assure l'intégrité des données :
  - Gestion de clés étrangères,
  - Intégrité de domaine,
  - Gestion de transactions,
  - Unicité
  - .....
- Interface d'accès via un langage de requêtes intégrant les opérateurs de l'algèbre relationnelle (ex SQL...)
- Optimisation des requêtes.

# BASES DE DONNÉES RELATIONNELLES.....

- Alors oui, les SGBDr s'occupent du stockage des données, MAIS PAS QUE !...
- Quelques SGBDr :
  - Oracle RDBMS,
  - IBM DB2,
  - Microsoft SQL Server,
  - Oracle MySQL,
  - Maria DB,
  - PostgreSQL

# COLLECTION DE DONNÉES ORGANISÉE EN TABLES

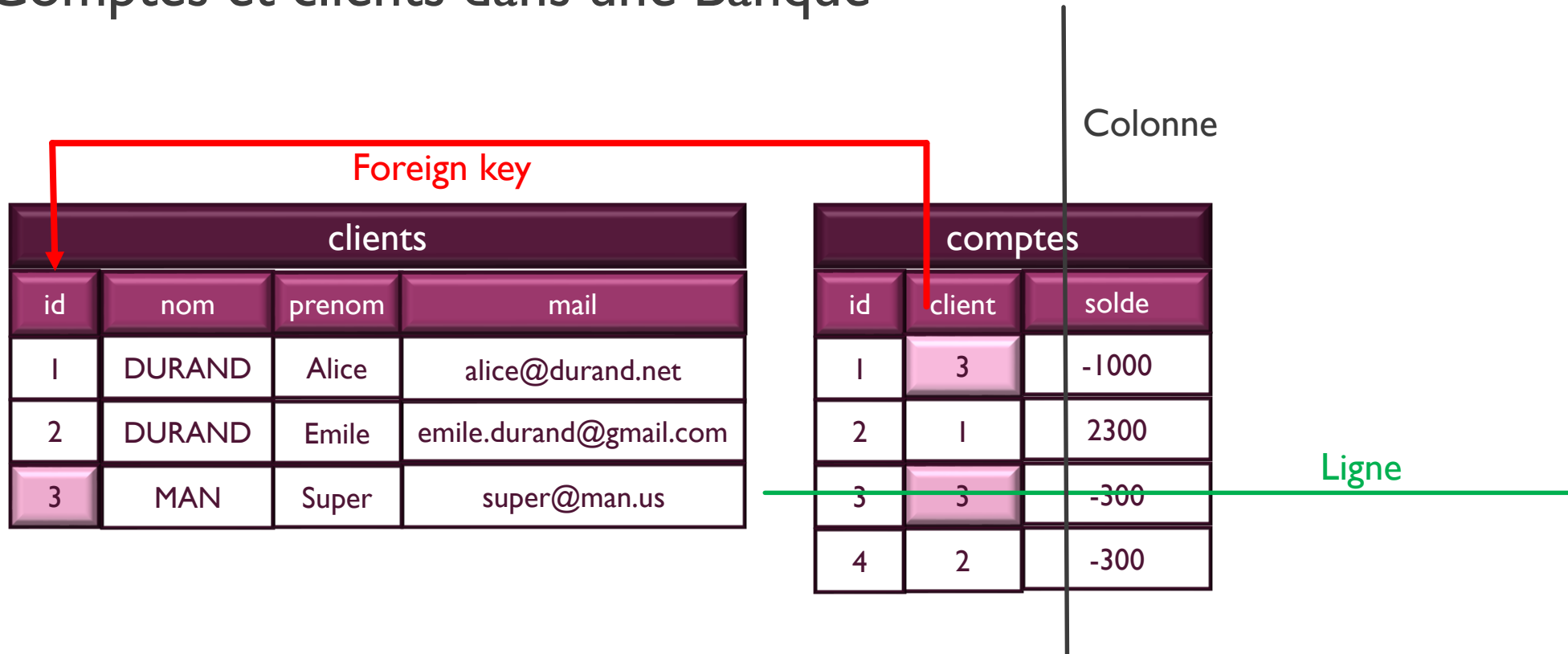
- L'organisation des tables doit répondre à des règles de normalisation que l'on appelle les formes normales
  - Il existe 6 niveaux de Formes Normales 1NF, 2NF,...6NF plus une forme normale, dite de BOYCE-CODD, qui intègre 2NF et 3NF et rend quelque peu caduque ces 2 dernières.
  - L'objectif du cours n'est pas de présenter formellement cette normalisation. La bible française sur le sujet est celle de François de Sainte Marie que l'on trouve ici :

<https://fsmrel.developpez.com/basesrelationnelles/normalisation>

- Existe en version PDF (En bas à droite de la première page)
- Relisez la bible une fois, deux fois, dix fois, il restera toujours quelque chose de nouveau à comprendre !!!

# EXEMPLES DE TABLES 'EN RELATIONS'

## ■ Comptes et clients dans une Banque





# LES LANGAGES DES SGBDR

- SQL - DDL (SQL - Data Definition Langage)
  - Définition des tables
    - CREATE TABLE, DROP TABLE, ALTER TABLE
  
- SQL – DML (SQL – Data Manipulation Langage)
  - Comme son nom l'indique : langage de manipulation de données
    - INSERT, UPDATE, DELETE, SELECT, START TRANSACTION/BEGIN, COMMIT, ROLLBACK, SAVEPOINT, ROLLBACK TO SAVEPOINT,...
  
- SQL – Stored Programs
  - Functions, Procedures, Triggers, Event Scheduler...

# BASES DE DONNÉES RELATIONNELLES (6)

## ■ DDL (MySQL) - 2

delimiter \$\$

```
DROP      TABLE IF EXISTS `it-akademy`.`clients`      $$
CREATE    TABLE      `it-akademy`.`clients`
(
    `id`      integer(10)      unsigned NOT NULL      AUTO_INCREMENT      ,
    `nom`     varchar(128)      NOT NULL      ,
    `prenom`  varchar(128)      NOT NULL      ,
    `email`   varchar(255)      NOT NULL      ,

    PRIMARY KEY (`id`)      ,
    KEY `clients_nom_prenom_idx` (`nom`,`prenom`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8      $$
```

# DDL (I)

## ■ DDL (MySQL) - I

delimiter \$\$

```
DROP TABLE IF EXISTS `it-akademy`.`comptes` $$
CREATE TABLE `it-akademy`.`comptes`
(
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT ,
    `clients` int(10) unsigned NOT NULL ,
    `solde` decimal(16,4) NOT NULL DEFAULT '0.0000' ,

    PRIMARY KEY (`id`) ,
    KEY `comptes_client_FK_idx` (`clients`) ,

    CONSTRAINT `comptes_client_FK`
        FOREIGN KEY (`clients`) REFERENCES `clients` (`id`)
        ON DELETE CASCADE
        ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8 $$
```

# DDL (2)

## ■ DDL (MySQL) - Outils

- MySQL Workbench - l'outil standard de référence
- phpMyAdmin - l'outil du monde PHP
- 

## ■ Exercice

- Créer la base it-akademy, comprenant les tables 'clients' et 'comptes' avec phpMyAdmin

# DML – LECTURE DE DONNÉES (I)

## ■ SELECT – Syntaxe (Sans jointure)

SELECT [ALL | DISTINCT | DISTINCTROW ] *select\_expr* [, *select\_expr* ...]

[FROM *table\_references*

[WHERE *where\_condition*]

[GROUP BY {*col\_name* | *expr* | *position*} [ASC | DESC], ...

[HAVING *where\_condition*]

[ORDER BY {*col\_name* | *expr* | *position*} [ASC | DESC], ...]

[LIMIT {[*offset*,] *row\_count* | *row\_count* OFFSET *offset*}]

[INTO OUTFILE '*file\_name*' [CHARACTER SET *charset\_name*] *export\_options* | INTO DUMPFILE '*file\_name*' |  
INTO *var\_name* [, *var\_name*]]

[FOR UPDATE | LOCK IN SHARE MODE]] ;

# DML LECTURE DE DONNÉES (2)

## ■ SELECT – Exemple (Sans jointure)

```
SELECT    nom, prenom, mail
FROM      it-akademy.clients
WHERE     nom = 'DURAND'
ORDER BY  prenom DESC;
```



nom	prenom	mail
DURAND	Emile	emile.durand@gmail.com
DURAND	Alice	alice@durand.net

31/10/2018

# DML – INSERTION DE DONNÉES (I)

## ■ INSERT – Exemple Syntaxe I

```
INSERT INTO tbl_name  
[(col_name [, col_name] ...)]  
{VALUES | VALUE} (value_list) [, (value_list)] ...  
  
[ON DUPLICATE KEY UPDATE assignment_list] ;
```

# DML – INSERTION DE DONNÉES (2)

## ■ INSERT – Exemple Syntaxe I (I)

```
INSERT INTO it-akademy.clients  
      (nom      , prenom      , email      )  
VALUE ('MOUSE' , 'Mickey'    , 'mickey.mouse@disney.com' ) ;
```



clients			
id	nom	prenom	mail
1	DURAND	Alice	alice@durand.net
2	DURAND	Emile	emile.durand@gmail.com
3	MAN	Super	super@man.us
4	MOUSE	Mickey	mickey.mouse@disney.com



# DML – INSERTION DE DONNÉES (3)

## ■ INSERT – Exemple Syntaxe I (2)

```
SELECT LAST_INSERT_ID() INTO @NewClient ; /* Récupération de la dernière id créée */
```

```
INSERT INTO it-akademy.comptes  
  (client ,solde )  
VALUE  (@NewClient ,20000000.00 ) ;
```



comptes		
id	client	solde
1	3	-1000.00
2	1	2300.00
3	3	-300.00
4	2	-300.00
5	4	20000000.00

# DML – SUPPRESSION DE DONNÉES (I)

- SUPPRESSION – Syntaxe simplifiée

```
DELETE FROM tbl_name  
[WHERE where_condition]  
[ORDER BY ...]  
[LIMIT row_count]          ;
```

# DML – INSERTION DE DONNÉES (3)

## ■ DELETE – Exemple Syntaxe simplifiée

DELETE FROM      it-akademy.clients  
WHERE              nom = 'DURAND' ;

Requête **ENSEMBLISTE** !



clients			
id	nom	prenom	mail
3	MAN	Super	super@man.us
4	MOUSE	Mickey	mickey.mouse@disney.com

comptes		
id	client	solde
1	3	-1000.00
3	3	-300.00
5	4	20000000.00

Suppression des  
comptes  
associés du fait  
de la foreign key

# TRANSACTIONS AND LOCKING STATEMENTS (I)

- Positionnement du problème – Cas d'un virement bancaire

compte Mickey MOUSE		
id	client	solde
5	4	20000000.00

Virement + 2000



Compte Super MAN		
id	client	solde
1	3	-1000

compte Mickey MOUSE		
id	client	solde
5	4	19998000.00

Les 2 maj ou rien



Compte Super MAN		
id	client	solde
1	3	1000

# TRANSACTIONS AND LOCKING STATEMENTS (2)

## ■ Propriétés ACID d'une transaction (I)

- **ACID** comme **A**tomicité, **C**ohérence, **I**solation, **D**urabilité

- **Atomicité**

Assure qu'une transaction se fait au complet ou pas du tout : si une partie d'une transaction ne peut être faite, il faut effacer toute trace de la transaction et remettre les données dans l'état où elles étaient avant la transaction. L'atomicité doit être respectée dans toutes situations, comme une panne d'électricité, une défaillance de l'ordinateur.

- **Cohérence**

Assure que chaque transaction amènera le système d'un état valide à un autre état valide. Tout changement à la base de données doit être valide selon toutes les règles définies, incluant mais non limitées aux contraintes d'intégrité, aux rollbacks en cascade, aux déclencheurs de base de données, et à toutes combinaisons d'événements.

# TRANSACTIONS AND LOCKING STATEMENTS (3)

## ■ Propriétés ACID d'une transaction (2)

### ■ Isolation

Toute transaction doit s'exécuter comme si elle était la seule sur le système. Aucune dépendance possible entre les transactions. La propriété d'isolation assure que l'exécution simultanée de transactions produit le même état que celui qui serait obtenu par l'exécution en série des transactions. Chaque transaction doit s'exécuter en isolation totale : si T1 et T2 s'exécutent simultanément, alors chacune doit demeurer indépendante de l'autre.

### ■ Durabilité

Assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'une panne d'électricité, d'une panne de l'ordinateur ou d'un autre problème. Par exemple, dans une base de données relationnelle, lorsqu'un groupe d'énoncés SQL a été exécuté, les résultats doivent être enregistrés de façon permanente, même dans le cas d'une panne immédiatement après l'exécution des énoncés.

# TRANSACTIONS AND LOCKING STATEMENTS (4)

## ■ Transaction – Syntaxe (cas général)

```
START TRANSACTION;  
SELECT .....  
FOR UPDATE..... ;
```

```
UPDATE....  
UPDATE...
```

```
INSERT.....
```

```
COMMIT ; /* En cas de succès */  
ROLLBACK ; /* En cas d'anomalie */
```

Sans déclaration explicite de Transaction, le SGBD se place en autocommit. C'est une euphémisme qui signifie que toutes les mises à jour, insertions, suppressions **sont validées une par une.**

En d'autres termes, en cas d'erreur, **on ne peut plus revenir** sur les modifications de données préalablement réalisées.

} Déverrouille les données



# PHP – INTERFACE AVEC UN SGBDR

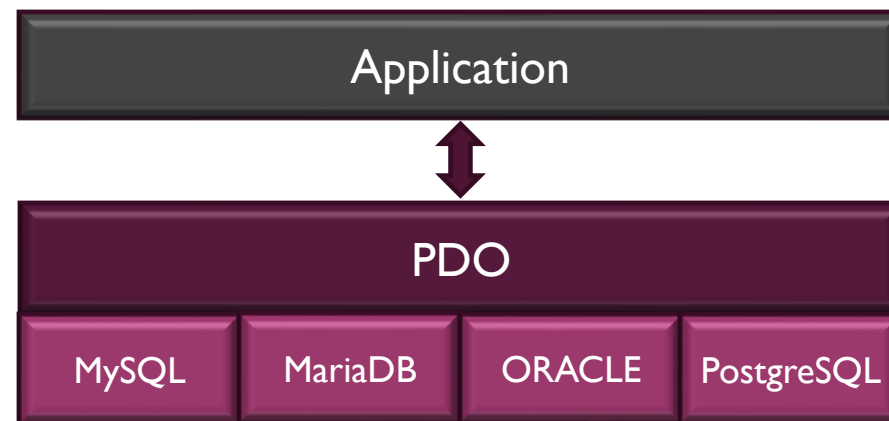
---

PDO



# PHP DATA OBJECTS (PDO)

- Interface OO d'abstraction au SGBDR (à partir de PHP 5.1)
  - Dans le cas de WAMP, à activer si nécessaire (menu PHP/Extensions)



# PDO – CONNEXION AU SGBDR

## ■ Syntaxe:

`$DataBase = new PDO('string $DSN [, string $username [, string $password [array $options ]]]) ;`

- **\$DSN - Data Source Name**, se décompose en pratique de la manière suivante :

- ❖ [Préfixe DSN][host:namehost][:port][;dbname=database\_name][;charset=charset\_name]

- ❖ Pour MySQL, le préfixe est '**mysql:**'

- ❖ Exemple : `$DSN = 'mysql:host=localhost;dbname=it-akademy;charset=utf8'`

- **\$username et \$password**

- Comme leur nom l'indique

# PDO – DÉCONNEXION DU SGBDR

- Syntaxe:

```
$DataBase = null ;
```

# PDO – FAIRE UNE REQUÊTE ÉLÉMENTAIRE - QUERY

## ■ Méthode : query (PDO::query)

Exécute une requête SQL en appelant une seule fonction, retourne le jeu de résultats (s'il y en a) de la requête en tant qu'objet PDOStatement.

**PDOStatement** : Représente une requête préparée et, une fois exécutée, le jeu de résultats associé.

- Syntaxe : `$query = $DataBase->query('La Requête à exécuter');`

Ex : `$clients = $DataBase->query('select nom, prenom, mail from clients  
where nom = 'DURAND' order by prenom DESC');` ;

- On récupère ainsi un jeu de résultats (par exemple plusieurs lignes car un moteur relationnel retourne un ensemble) qu'il faut aller chercher un à un via la méthode `PDOStatement::fetch`

# PDO – FAIRE UNE REQUÊTE ÉLÉMENTAIRE - FETCH

## ■ Méthode : query (PDOStatement::fetch)

Récupère la ligne suivante d'un jeu de résultats PDO

- Syntaxe simple : `$fetch = $query->fetch()` ;
  - ❖ `$fetch` est un array qui contient l'ensemble des champs de la première ligne de réponse
  - ❖ Ex :

```
$client    = $clients->fetch() ;
$nom       = $client['nom']     ;
$prenom    = $client['prenom'] ;
$email     = $client['email']  ;
```
  - ❖ En pratique, on récupère toutes les lignes au moyen d'une boucle `while()` contrôlée de la manière suivante :

```
while($client = $clients->fetch())
{....}
```
  - ❖ En fin de boucle, on ferme le curseur : `$clients->closeCursor()` ; /\* PDOStatement::closeCursor \*/

# PDO – EXERCICE QUERY

- A partir d'un formulaire ou l'on saisit le nom, afficher tous les clients qui porte ce nom saisi.
- Pour gérer les erreurs d'accès à la base de données, on utilisera la gestion d'exceptions au moyens des blocs try{} et catch{}

```
try
{
    /*
    **  Gestion accès via PDO à la base de données
    */
}
catch(Exception $Exception)
{
    die($Exception->getMessage())      ;    /* arrête le script en envoyant le message d'erreur détectée */
}
```

# PDO –REQUÊTES PRÉPARÉES – PREPARE/EXECUTE (I)

- Méthodes : PDO::prepare() et PDOStatement::execute()
  - PDO::prepare()  
Prépare une requête SQL à être exécutée par la méthode PDOStatement::execute(). La requête SQL peut contenir zéro ou plusieurs noms (:nom) ou marqueurs (?) pour lesquels les valeurs réelles seront substituées lorsque la requête sera exécutée.
- PDOStatement::execute()
  - Exécute une requête préparée. Si la requête préparée inclut des marqueurs de positionnement, on utilisera pour passer les arguments en entrée :
    - Soit la méthode PDOStatement::bindParam() (or PDOStatement::bindValue()),
    - Soit un tableau de valeurs de paramètres
- PDOStatement::nextRowset()
  - Avance à la prochaine ligne de résultats (utilisé lorsque la requête contient plusieurs ordres SQL ou si une procédure stockée renvoie plusieurs lignes).

# PDO –REQUÊTES PRÉPARÉES – PREPARE/EXECUTE (2)

## ■ Exemple(1)

```
/*
**      (1)      Prepare
*/
$sql_request = $database->prepare ( '      INSERT INTO comptes
                                     (client ,      solde)
                                     VALUES      (:client      ,      :solde);
                                     )
                                     ;

/*
**      (2)      Bind
*/
$sql_request->bindParam (':client'      ,      $ClientID      ,      PDO::PARAM_INT)      /* paramètre entier */      ;
$sql_request->bindParam (':solde'      ,      $Solde      ,      PDO::PARAM_STR)      /* paramètre string */      ;

/*
**      (3)      Execute
*/
$sql->execute()
```



# PDO –REQUÊTES PRÉPARÉES – PREPARE/EXECUTE (3)

## ■ Exemple(2) – variante array pour passer les paramètres en entrée

```
/*  
** (2) Pas de bindParam() et execute() direct  
*/  
$sql->execute ( array (  
    'client' => $ClientID ,  
    'solde' => $Solde  
)  
);
```

## PDO – EXERCICE REQUÊTE PRÉPARÉE

- A partir d'un formulaire ou l'on saisit le nom, prénom et email, insérer le clients ainsi saisi.

# PDO –GESTION DE TRANSACTIONS – MISE EN ŒUVRE

- Ecrire un formulaire de virement qui débite le 1<sup>er</sup> compte d'un client et crédite le 1<sup>er</sup> compte d'un autre client
  - On saisira les noms et prénoms des clients ainsi que le montant du virement

# PDO –GESTION DE TRANSACTIONS - EXERCICE

## ■ Méthodes

- PDO::beginTransaction()
- PDO::commit() ou PDO::rollback()