Web dynamique avec php et mySQL

F. Tort & J. Vétois

ENS de Cachan

14, 15, 16 juin 2006

1 Les principes du Web dynamique

Web classique = des pages définitives, codées en HTML, téléchargé telles que par le visiteur.
Web dynamique = des pages, codées dans un langage générant du HTML, se créent à la volée, selon les caractéristiques de la demande du visiteur.

A. Architecture client serveur du Web classique

Les pages existent sur le serveur et son téléchargées telles que par le client.

B. Web dynamique côté client

Le client télécharge un fichier codé dans un langage générant du HTML. La création de la page se réalise chez le client.
Exemple : javaScript.

C. Web dynamique côté serveur

Le serveur dispose d'un fichier codé dans un langage générant du HTML, lors de la demande de téléchargement du client, il construit la page HTML, éventuellement en utilisant des paramètres envoyés par le client.
Exemple : programme CGI (interfacé avec C, Perl, etc.), programme PHP.
Avantages de PHP :

-facilité de passage de paramètres via un formulaire (des variables désignant les champs du formulaire sont utilisables directement dans un programme PHP)

-intégration avec HTML qui permet dans un même fichier d'écrire dans les deux langages (HTML pour les parties fixes,, PHP pour les parties dynamiques)

-facilité de la gestion de session : PHP offre des fonctions permettant de mémoriser les informations relatives à plusieurs demandes d'un même visiteur.

D. Avec base de données

Une base de données est un ensemble de fichiers conservant une grande quantité de données structurées. Ces données diffèrent des variables d'un programme par leur persistance en mémoire. Il existe différentes façons de structurer ces données, la plus répandue est le modèle relationnel.
La gestion d'une base de données nécessite des fonctions de création de la base, de mise à jour des données, d'accès aux données (requête), et d'administration des accès et des utilisateurs.

Un standard existe pour le relationnel : le langage SQL (Structured Query Langage). MySql est un langage de programmation respectant pour partie ce standard.

MySQL comporte :

-le serveur *mysqld*. Chargé d'accéder aux fichiers stockant les données, pour lire et écrire des informations.

-les utilitaires chargés de dialoguer avec le serveur, par l'intermédiaire d'une connexion, pour accomplir un type de tâche particulier. Parmi eux, *mysql* permet d'envoyer des commandes directement au serveur.

-il est possible de créer un client *mysql* à partir d'un programme PHP, à l'aide d'un ensemble de fonctions spécifiques (appelé API, Application Programming Interface)

2 Bases de données relationnelles

On considère généralement qu'il convient de dissocier la description conceptuelle d'une BD de son organisation informatique. Le groupe de normalisation américain ANSI-SPARC a proposé 3 niveaux :

-**niveau conceptuel** : dans un schéma sont décrites les entités du monde réel indépendamment de toute technique d'organisation et d'implantation des données , à ce niveau on utilise généralement un modèle sémantique.

Exemple : schéma Entité Association, UML.

-**niveau logique** : description de l'organisation de la BD permettant de faciliter la manipulation des données (regroupement de données, chemin d'accès, etc.).

Exemple : le modèle relationnel

-**niveau physique** : spécification du mode d'implantation des structures de données en mémoire auxiliaires conformément aux fonctionalités offertes par le logiciel de gestion de base de données.

Exemple : la norme SQL, ORACLE, mySql.

On étudiera ici le modèle relationnel (niveau logique de description) et le langage mySql (niveau physique de description).

2.1 Modèle relationnel des données

A. Principes: la notion de relations

Il s'agit de regrouper les informations élémentaires à mémoriser, par individus, et même par ensemble d'individus.

Données élémentaires : une "valeur" non décomposable.

Exemple : 3, rouge, 12/03/1961.

Individu : un ensemble de valeurs.

Exemple : Durand, Philippe, 1961, 3 rue des Gâtines.

Relation : un ensemble d'individus.

Exemple : (Durand, Philippe, 1961, 3 rue des Gâtines ;
Dupond, Juliette, 1954, 34 allée des fusains).

Base de données : un ensemble de relations.

B. Représentation d'une relation : une table

Les individus en lignes, une seule valeur dans chaque cellule.

nom	prénom	annéeNaiss	adresse
Durand	Philippe	1961	3, rue des Gâtines
Dupond	Juliette	1954	34, allée des Fusains

C. Notation d'une relation : le schéma de la relation

Une relation est notée par son nom, et une liste de champs.

Exemple : PERSONNE(nom, prénom, annéeNaiss, adresse)

Attribut : caractéristique prenant une valeur, d'un certain type, pour chaque individu.

Clé (identifiant) : l'un des attributs joue le rôle particulier d'identifier de façon unique un individu. La valeur prise par cet attribut pour un individu est unique dans la relation.

Notation : nom attribut

Exemple : PERSONNE(id pers, nom, prénom, annéeNaiss, adresse)

D. sémantique du modèle : **les dépendances fonctionnelles**

Définition : Deux attributs a et b sont reliés par une dépendance fonctionnelle, dans le sens b dépend fonctionnellement de a, si à une valeur quelconque de a, on ne peut faire correspondre qu'une et une seule valeur de b, dans le modèle.

Notation : a --*df*--> b

Exemple : *un "'id personne" ne correspond qu'à un seul nom. Intuitivement : un code identifie une personne, dans le modèle une personne n'a qu'un nom.*

Propriétés :

Transitivité : Si a --*df*--> b et b--*df*--> c, alors a --*df*--> c,
où a est un attribut ou un groupe d'attributs, b est un attribut, c est un attribut

Augmentation : Si a --*df*--> b alors a + c --*df*--> b,
où a et c sont des attributs ou groupes d'attributs, b est un attribut .

Df et schéma de relation :

Un schéma de relation exprime les dépendances fonctionnelles suivantes :

NOM_REL(clé, attribut 1, attribut 2) ↔ clé --*df*--> attribut 1, clé --*df*--> attribut 2

2.2 Schéma relationnel normalisé

Pour un cas donné, des modèles relationnels différents peuvent être conçus. Lequel choisir ? A partir de quelques problèmes de manipulation de données, certaines propriétés d'un bon schéma ont été établies, qui fondent la **théorie de la normalisation** des bases de données.

Trois types d'anomalies de manipulation existent :

-anomalie d'insertion,

-anomalie de modification,

-anomalie de suppression.

Afin de réduire ces anomalies, il est préférable de construire un modèle de base de données en **troisième forme normale**, tel que :

-chaque relation possède une clé

-dans chaque relation les attributs dépendent fonctionnellement de la clé

-il n'existe pas de df entre attributs hors clé

-les df exprimées (de la clé vers les attributs) sont toutes élémentaires

-les df exprimées (de la clé vers les attributs) sont toutes directes

Df directe : non construite par transitivité d'autres df.
Df élémentaire : non construite par augmentation d'une autre df.

Appliquer ces règles de construction revient à s'efforcer de répartir les informations entre plusieurs relations différentes, plutôt que de les réunir dans un nombre réduit de relations.

2.3 Associations entre relation dans un schéma normalisé

A. les clés étrangères

On appelle clé étrangère la clé d'une relation lorsqu'elle figure parmi les attributs d'une autre relation. C'est un outil permettant de définir des associations entre les relations, tout en respectant les règles de normalisation.

Notation : #nomCléEtrangère

B. Association un à plusieurs

Lorsque tout individu d'une relation R est lié à un seul individu d'une relation R', la clé de la relation R' est attribut de la relation R (et y est appelée clé étrangère "hors identifiant")

Exemple : PERSONNE(id pers, nom prénom, annéeNaiss, adresse, #ville)
LIEU(nom ville, département, pays)

Supposons que la relation LIEU soit définie dans la base. La contrainte : une personne vit dans une ville, et une seule, se traduit par l'ajout de la clé étrangère #ville dans la relation PERSONNE.

C. Association plusieurs à plusieurs

Lorsque tout individu d'une relation R est lié à un ou plusieurs individu d'une relation R', une relation R" est créée, dont la clé est formée des deux clés de R et R' (appelées clés étrangères).

Exemple : ACHAT(#id pers, #réf produit)

PRODUIT(réf produit, prix)

Supposons que la relation PRODUIT soit définie dans la base. La contrainte : une personne peut acheter un ou plusieurs produit, se traduit par la création d'un e relation ACHAT identifiée par le code de la personne et la référence du produit.

Une relation identifiée par des clés étrangères peut elle-même posséder des caractéristiques propres, qui seront autant d'attributs de la relation.

Exemple : ACHAT(#id pers, #réf produit, quantité achetée)

La quantité de produit achetée par un individu caractérise un achat.

D. Cas particuliers d'associations.

Association transitive : Une relation peut être en relation avec elle-même.

-si c'est une relation de un à plusieurs, elle comporte sa propre clé comme clé étrangère,

Exemple : chaque personne a un supérieur hiérarchique, figurant parmi les personnes.

PERSONNE(id pers, nom, prénom, annéeNaiss, adresse, #id chef)

-si c'est une relation de plusieurs à plusieurs, une relation identifiée par une clé constituée deux fois de sa clé est créée.

Exemple : chaque personnes a plusieurs supérieurs hiérarchiques, figurant parmi les personnes.

PERSONNE(id pers, nom, prénom, annéeNaiss, adresse)

HIERARCHIE(#id pers, #id supérieur)

Association n-aire : Il est possible de décrire des associations plusieurs à plusieurs impliquant plus de deux relations. On parle d'association n-aire, les plus courantes étant ternaires. Dans ce cas, une relation est créée identifiée par l'ensemble des clés de chaque relation associée. Elle peut posséder des attributs propres.

Exemple : *une personne achète un produit à un magasin à une date donnée, en une quantité donnée.*.

MAGASIN(id magasin, adresse Mag , tél Mag, #ville)

ACHAT(#id pers, #réf produit, #id magasin, dateAchat, quantité)

La date participe à l'identifiant selon que l'on mémorise l'historique des achats ou non.

Récapitulation sur les dépendances fonctionnelles dans un modèle normalisé:

NOM_REL(clé, attribut 1, attribut 2) ↔ clé --*df*--> attribut 1,
clé --*df*--> attribut 2
Et donc :
NOM_REL(clé, …, #clé_étrangère) ↔ clé --*df*--> clé_étrangère
Mais aussi :
NOM_REL(#clé_étrg1, #clé_étrg2, attribut)
↔ clé_étrg1, clé_étrg2 --*df*--> Aattribut
… et aucune autre.

2.4 Algèbre relationnelle

Tout l'intérêt d'une base de données, outre la mémorisation de données persistantes, est la possibilité de l'interroger ; c'est à dire d'en extraire des informations respectant certains critères.

On dispose pour cela d'un ensemble d'opérateurs permettant de construire de nouvelles relations à partir des relations de la base.

$$Op(R) = R'$$

$$Op(Op'(R)) = R''$$

A. Opérateurs relationnels

Projection	$R' = \pi_{attributs}\ R$
Conserve certains attributs d'une relation R.	

Schéma de R' = *attributs,* individus de R' = individus de R

Sélection	$R' = \sigma_{(critère)}\ R$
Conserve les individus d'une relation R respectant un critère sur les valeurs des champs.	

Critère : expression logique sur 'attribut *op* constante', où op = opérateur de comparaison.

Opérateur logiques : et, ou, non.

Schéma de R' = schéma de R

Jointure	$R' = R1 \ltimes_{R1.attribut1 = R2.attribut2} R2$
Combine les individus de deux relations	

qui prennent la même valeur pour deux attributs donnés.

Les deux attributs doivent être de même type.

Schéma de R' = schéma de R1 ∪ schéma de R2

Requête (exemple)

Liste des coordonnées des parisiens nés avant 2000.

$$\pi_{\text{nom, prénom,adresse}} (\sigma_{\text{(pays='Paris" et annéeNaiss < 2000)}}(\text{PERSONNE} \succ\!\prec_{\text{nomVille}} \text{LIEU}))$$

B. Opérateurs ensemblistes

Ces opérateurs sont binaires, ils s'appliquent à des relations uni-compatibles.

Schéma de R' = schéma de R1 = schéma de R2

Union	R' = R1 union R2

Individus de R' = .individus de R1 ∪ individus de R2

Intersection	R' = R1 inter R2

Individus de R' = .individus communs à R1 et R2

Elle peut être traduite par une jointure.

Différence	R' = R1 except R2

Individus de R' = .individus de R1 pas dans R2

Elle permet d'exprimer la négation : en retenant les individus de R1 qui n'ont pas une propriété liées aux individus de R2.

Requête (exemples)

Liste des villes où habitent des personnes qui ont des magasins.

$$\pi_{\text{ville}} (\text{PERSONNE}) \text{ INTER } \pi_{\text{ville}} (\text{MAGASIN})$$

Liste des villes (où habitent des personnes) qui n'ont pas de magasin.

$$\pi_{\text{ville}} (\text{PERSONNE}) \text{ EXCEPT } \pi_{\text{ville}} (\text{MAGASIN})$$

Nom des personnes qui n'habitent pas à Paris.

$$\pi_{\text{nom}} (\text{PERSONNE}) \text{ EXCEPT } \pi_{\text{nom}} ((\sigma_{\text{(ville='Paris')}}(\text{PERSONNE}))$$

ANNEXE : la base exemple

PERSONNE(id pers, nom, prénom, annéeNaiss, adressePer, #ville)

HIERARCHIE(#id pers, #id supérieur)

LIEU(nom ville, département, pays)

ACHAT(#id pers, #réf produit, #id magasin, dateAchat, quantité)

PRODUIT(réf produit, prix)

MAGASIN(id magasin, adresse Mag , tél Mag, #ville)

3. Créer et utiliser une base de données avec à mySQL

SQL (*Structured Query Language*) est le langage de gestion des bases de données relationnel, devenu une norme ANSI en 1988 (SQL-1), révisé en 1992 (SQL-2).

MySQL est un langage de gestion de base de données relationnel conçu pour le développement de site Web dynamique. Il applique en partie et étend la norme SQL ANSI.

Attention au changement de vocabulaire.

Relation	Table
Attribut	Champ
Individu	Enregistrement, ligne

3.1 Construire une base

A. Créer une nouvelle base

La base est créée sur le serveur mySQL par son administrateur. Si l'utilisateur de la base est différent de l'administrateur, cet utilisateur doit avoir un compte sur le serveur, et les droits sur la base doivent lui être attribués. A chaque fois que l'utilisateur de la base désire y accéder, il doit s'identifier et s'authentifier auprès du serveur.

Instructions	Description
CREATE DATABASE *nomBase* ;	Crée une nouvelle base de donnée
GRANT ALL PRIVILEGES ON *nomBase*.* TO *loginUtilisateur* ;	Attribue tous les droits de manipulations de la base à un utilisateur. *nomBase*.* désigne tout le contenu de la base

B. Ajouter une table

Avant toute manipulation, il faut désigner la base de données utilisée.

Instructions	Description
USE *nomBase* ;	Désigne la base de donnée concernée

Pour créer une nouvelle table, il faut indiquer son nom, la liste de ses champs, et pour chacun son type de données et sa taille. Sous UNIX, mySQL distingue majuscule et minuscule dans les noms des tables.

Instructions	Description
CREATE TABLE *nomTable* (*nomChamp TYPE CONTRAINTE*, …); Voir liste ci après	Ajoute une table. *nomTable, nomChamp,* peuvent comporter des accents, mais à éviter. *TYPE* est obligatoire. *CONTRAINTE* est optionnelle, on peut en combiner plusieurs.
TYPE = FLOAT, DOUBLE CHAR(*M*), VARCHAR (*M*)	 Réel sur 4 et 8 octets Chaîne de M caractères, figée pour CHAR

SMALLINT, INTEGER, BIGINT	Entier, sur 2, 4 et 8 octets
DATE, TIME, DATETIME	Formats : aaaa-mm-jj, hh :mm :ss, les deux
YEAR[2|4]	Année sur 2 ou 4 chiffres, à préciser.
TIMESTAMP	Permet d'estampiller à la date de modification
ENUM('*val1*', '*val2*', …)	Type énuméré, valeurs précisées dans les ()
CONTRAINTE =	
DEFAULT *valeur*	Désigne une valeur prise par défaut
NOT NULL	Interdit l'absence de valeur
UNIQUE	La valeur est unique pour ce champ sur l'ensemble des enregistrements de la table
PRIMARY KEY	Définit un index qui sert de clé à la table.
Pour définir une clé, il faut ajouter les deux conditions : UNIQUE et NOT NULL	
FOREIGN KEY	Désigne une clé étrangère

Exemple :

```
CREATE TABLE PERSONNE(
        idPers INTEGER NOT NULL UNIQUE,
        nom VARCHAR(20),
        prénom VARCHAR(20),
        adressePers VARCHAR(50),
        anneeNaiss INTEGER) ;
```

C. Modifier une table

Instructions	Description
ALTER TABLE *nomTable* ADD *nomChamp1* *TYPE CONTRAINTE* ;	Ajoute un champ à la table. La valeur sera à NULL où à la valeur par défaut si des enregistrements existent déjà dans la table.
ALTER TABLE *nomTable* DROP *nomChamp* ;	Détruit le champ.
ALTER TABLE *nomTable* MODIFY *nomChamp TYPE ;*	Modifie le type d'un champ.

Exemple :

```
ALTER TABLE PERSONNE ADD ville VARCHAR(20) ;
```

C. Effacer une table

Instructions	Description
DROP TABLE *nomTable* ;	Efface la table. Les données sont perdues.

D. Voir la structure de la base

Instructions	Description
SHOW TABLES ;	Affiche la liste des tables de la base
DESCRIBE *nomTable* ;	Affiche le schéma de la table.

3.2. Mettre à jour une base

A. Insérer de nouveaux enregistrements

Instructions	Description
INSERT INTO *nomTable* (*nomChamp1, nomchamp2,...)*	Ajoute un enregistrement à la table, dont les valeurs sont données explicitement.

VALUES (*valeur1, valeur2*) ;	
INSERT INTO *nomTable* (*nomChamp1, nomchamp2,...)* SELECT … ;	Ajoute un enregistrement à la table, dont les valeurs sont issues d'une requête sur la base.
LOAD DATA INFILE '*fichier.txt*' INTO TABLE *nomTable ;*	Importe le fichier .txt dans la table. Ce fichier doit comporter un enregistrement par ligne, et espacer les valeurs par des tabulations.

Exemple :

```
INSERT INTO PERSONNE(
        idPers, nom, prénom, adressePers, anneeNaiss)
        VALUES (1, Durand, Philippe, 3 rue Gâtines, 1965) ;
```

B. Modifier des enregistrements

Instructions	Description
UPDATE *nomTable* SET *nomChamp = 'valeur',* *nomChamp = 'valeur'* WHERE *condition;*	Modifie tous les enregistrements respectant la condition, en remplaçant les champs indiqués par les valeurs indiquées. *Condition : nomChamp = 'valeurs'*

C. Supprimer des enregistrements

Instructions	Description
DELETE FROM *nomTable* WHERE *condition;*	Détruit tous les enregistrements respectant la condition.

Exemple :

```
UPDATE PERSONNE SET anneeNaiss = 1962
        WHERE idPers = 1 ;
```

3.3 Interroger une base

A. Requêtes simples

SFW

Instructions	Description
SELECT * | *liste champs* FROM *nomTable* [WHERE *condition*]*;*	Retient de la table désignée par FROM les enregistrements respectant la condition du WHERE, en affichant tous les champs (*) où seulement ceux désignés par SELECT.

Exemples :

```
SELECT nom, prenom FROM PERSONNE WHERE ville = 'Paris';

SELECT réfProduit FROM PRODUIT WHERE prix > 300 ;
```

Pour **afficher le contenu d'une table** R : SELECT * FROM *R* ;

Clause select

Instructions	Description
SELECT DISTINCT … FROM … *;*	Elimine les doublons dans la table résultat.

SELECT *champ* AS *nom* FROM … *;*	Renomme l'attribut par *nom* dans la table résultat.
SELECT *FONCTION*(*champs)* FROM … *;* Pas de blanc entre le nom de la fonction et la parenthèse ouvrante	Applique la fonction aux champs à chaque enregistrement..
FONCTION = - + * / LENGTH(*champ*) CONCAT(*liste champs*) SUBSTRING(*champs, n, k*) YEAR(*champ*) NOW() DATE_ADD(*date, champ*)	 Opérations arithmétiques, champs numériques. Longueur de la valeur du champ Concatène les valeurs de champs VARCHAR Sous chaîne de la nième à la kième lettre Extrait l'année d'un champ DATE Donne la date du jour. Additionne des dates ou des champs DATE
Ces calculs, appliqués à une valeur NULL donnent NULL.	
SELECT '*constante*' FROM … ;	Affichera la constante pour chaque enregistrement, combinable avec CONCAT.

Exemples :

```
SELECT CONCAT ('Monsieur ', nom, ' ' , prenom) FROM PERSONNE ;
SELECT nom, YEAR(NOW()) – anneeNaiss AS age FROM PERSONNE ;
```

Clause Where

La condition du WHERE correspond à un ensemble de critères que doivent respecter les champs des enregistrements de la table.

Instructions	Description
WHERE *champ OPERATEUR valeur* *OPERATEUR :* = | < | > | <= | >= | <>	La condition la plus simple est un critère sur un champ de la table.
WHERE *cov* AND | OR *cov* WHERE NOT (*cov*)	Conjonctions ou disjonctions de critères. Les parenthèses permettent de changer la priorité.
WHERE *champ* BETWEEN *valeur1* AND *valeur2*	Comparaison à un intervalle.
WHERE *champ* IN | NOT IN (*liste valeurs)*	Comparaison à un ensemble de valeurs.
WHERE *champ* NOT LIKE | LIKE (*chaîne)*	Comparaison à un filtre. _ pour n'importe quel caractère, % pour n'importe quelle chaîne de caractères
WHERE *champ1 = champ2*	Comparaison à un autre champ.

Exemples :

```
SELECT idPers FROM PERSONNE WHERE ville NOT IN ('Paris', 'Nice') ;
SELECT idPers FROM PERSONNE WHERE ville LIKE '_a%' ;
SELECT idPers, refProduit FROM ACHAT WHERE date = '2001-03-10' ;
```

ORDER BY

On peut trier les enregistrements de la table résultat.

Instructions	Description
SELECT … FROM … [WHERE …] ORDER BY *liste champs* [DESC]*;*	Trie les enregistrements dans l'ordre croissant (décroissant si DESC) des valeurs prises par les champs indiqués.

SELECT FONCTION (*champs*) AS *nom* FROM … [WHERE …] ORDER BY *nom;*	Tri sur les valeurs d'un champ calculé, nécessité de nommer (avec AS) ce champs.

Exemple :

```
SELECT nom, prenom FROM PERSONNE WHERE ville ='Paris' ORDER BY nom ;
```

Valeur NULL

La valeur NULL est une absence de valeur, on ne peut donc lui appliquer aucune opération ou comparaison usuelle. De plus, NULL n'est pas une chaîne de caractère, mais un mot-clé.

Instruction	Description
SELECT … FROM … WHERE *champs* IS NULL | IS NOT NULL*;*	Teste l'absence de valeur (la valeur NULL) sur un champ.
SELECT IFNULL(*champ*, '*constante*') FROM … WHERE … ;	Donne la valeur *constante* aux champs de valeur NULL.

Pièges courants :

-ne pas tester : ~~*champs* = NULL~~

-SELECT * FROM PRODUIT WHERE prix < = 20 AND prix >= 20 ; ne retournera pas les produits qui n'ont pas de prix.

B. Sélection sur plusieurs tables

Jointures

Instruction	Description
SELECT *liste de champs* FROM *table1, table2* WHERE *table1.champ1 = table2.champ2* ;	Réalise la jointure de table1 et table2, sur les champs précisés dans le WHERE.
Pour lever les ambiguïtés sur les noms des champs, on utilise une notation pointée avec le nom de la table : table.champ	
SELECT *liste de champs* FROM *table1 alias1, table2 alias2* WHERE *alias1.champ1 = alias2.champ2*;	On peut renommer les tables par des alias plus courts, pour faciliter l'écriture de la condition de jointure.

Exemple :

```
SELECT prix, quantité
FROM PERSONNE P, ACHAT A, PRODUIT PR
WHERE P.idPers = A.IdPers
    AND A. refProduit = PR.refProduit
    AND idPers = 23 AND idMagasin = 4 ;
```

Auto jointure

Pour joindre une table avec elle-même on doit utiliser le renommage de la table, pour lever l'ambiguïté sur les champs utilisés.

Instruction	Description
SELECT *alias1.champ1, alias2.champ2*	Réalise une jointure d'une table avec elle-

FROM *table1 alias1*, *table1 alias2* WHERE *alias1.champ3 = alias2.champ4* ;	même.

Exemple :

```
SELECT m1.nom, m2.nom
FROM MAGASIN m1, MAGASIN m2
WHERE m1.idMagasin = m2.idMagasin ;
```

Jointures externes

La jointure précédemment présentée ignore les enregistrements ne respectant pas le critère de jointure. Si l'on souhaite conserver tous les enregistrements de l'une des tables jointes, y compris ceux ne respectant pas le critère de jointure, on peut réaliser une jointure externe.

Instruction	Description
SELECT ... FROM *table1* NATURAL LEFT JOIN *table2* WHERE *table1.champ1 = table2.champ2* ;	Réalise une jointure dans laquelle les enregistrements de *table1* ne respectant pas le critère de jointure sont conservés et complétés de valeur NULL pour les champs de *table2*.

Opérations ensemblistes

MySQL ignore les opérateurs ensemblistes. Une intersection sera donc exprimée par une jointure. Une différence sera exprimée par une requête imbriquée (Voir *infra*).

C. Partition et agrégation

On peut réaliser des calculs portant sur plusieurs enregistrements, on dispose pour cela d'instructions permettant de partitionner les tables et d'exprimer des conditions sur ces partitions, et de fonctions d'agrégation

GROUP BY

Instruction	Description
SELECT ... FROM ... [WHERE ...] GROUP BY *liste champs* ;	Réalise une partition sur la table résultat selon les valeurs prises par les champs indiqués.
SELECT *champ1, AGREGAT*(*champ*2)… FROM … [WHERE ...] GROUP BY *champ1* ;	Calcule un agrégat sur *champ2* pour chaque partition sur *champ1*.
Règle : tout champ présent dans un SELECT avec un agrégat doit nécessairement faire l'objet d'un GROUP BY. (En fait, mySQL n'affiche pas d'erreur, et affiche une des valeurs trouvées).	
AGREGAT = COUNT | AVG | MIN | MAX | SUM | STD	Dénombrement, moyenne, minimum, maximum, somme, écart type.

Exemples :

```
SELECT AVG(prix) FROM PRODUIT ;

SELECT idPers, AVG(prix) FROM PERSONNE, ACHAT, PRODUIT WHERE ….
GROUP BY idPers ;
```

HAVING

Instruction	Description
SELECT ... FROM ... [WHERE ...] GROUP BY *liste champs* HAVING *condition* ;	Sélectionne les partitions respectant la condition.
La condition doit porter sur l'ensemble d'une partition, elle porte donc sur une agrégation	

Exemples :

```
SELECT idPers, AVG(prix) FROM PERSONNE, ACHAT, PRODUIT WHERE ….
GROUP BY idPers
HAVING AVG(prix)>300 ;
```

4. Introduction à la programmation en PHP

V. Polycopié « le langage php ».

5. PHP et mySQL pour un site web dynamique

5.1 Formulaires et programme PHP

A. Ecriture de formulaire avec HTML.

Les formulaires sont des pages Web permettant à l'utilisateur de donner des éléments spécifiques complétant sa demande par l'intermédiaire de champs de saisie, et de transmettre ses éléments par des boutons de soumission.

HTML comporte des balises permettant de créer de tels formulaires.

Balises	Description
<FORM> </FORM>	Délimite le formulaire. Peut contenir outre les champs de saisie, n'importe quel texte ou balise HTML.
Les options de <FORM> :	
ACTION = *programme.php*	Référence le programme qui doit être exécuté lors de la soumission du formulaire.
METHOD = GET | POST	Mode de transmission des données saisies : avec l'URL | à part.
ENCTYPE = - application/x-www-form-urlencoded - multipart/form-data	Type d'encodage des données du formulaire utilisé lors de la transmission : -dans l'URL sous la form nom=valeur séparés par des & -transmission séparée des fichiers
<INPUT TYPE = *type* **NAME** = *nom* **> </INPUT>**	Insère un champ nommé, dont l'apparence dépendra du type.
TYPE = **TEXT** SIZE = *taille*	Champ de saisie.
TYPE = **PASSWORD**	Champ de saisie dont le texte n'apparaît pas en clair (remplacé par des *), et est effacé à chaque rechargement de la page.
TYPE = **HIDDEN** VALUE = *valeur*	Champ caché. Permet de passer un paramètre de valeur fixée.
TYPE = **CHECKBOX** VALUE = *valeur*	Des boutons cochables associés à des libellés. VALUE n'est pas le libellé, mais la valeur transmise au serveur si la case est cochée. **Choix non exclusif**.
TYPE = **RADIO** VALUE = *valeur* CHECKED	Des boutons radio associés à des libellés. CHECKED indique éventuellement un choix sélectionné par défaut. **Choix exclusif**.
TYPE = **SUBMIT** VALUE=valeur	Bouton validant la saisie du formulaire, et déclenchant l'action ACTION. VALUE définit le libellé du bouton.

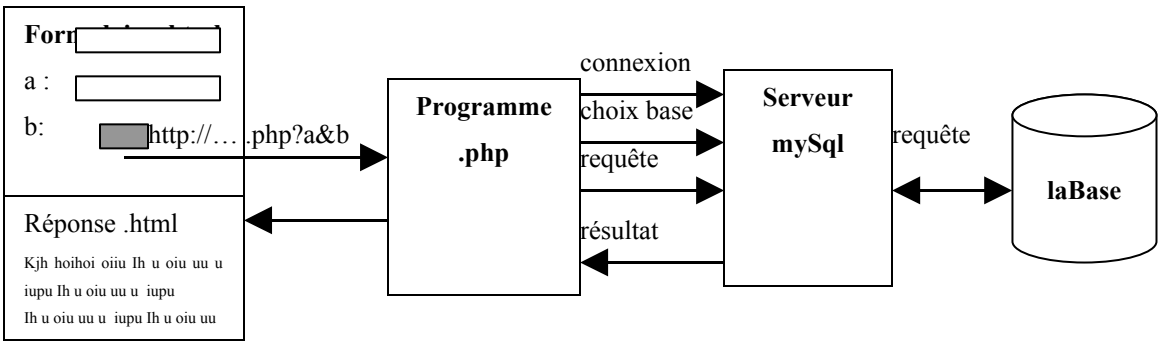
TYPE = **RESET** VALUE=valeur	Bouton demandant de réinitialiser le formulaire.
<SELECT NAME=nom SIZE=taille> **<OPTION** VALUE=valeur> libellé du choix <OPTION VALUE=valeur SELECTED> … **</SELECT>**	Affiche une liste d'options parmi lesquelles l'utilisateur peut faire un ou plusieurs choix SIZE définit le nombre de choix affiché simultanément, visibles sans dérouler la liste. C'est un conteneur dans lequel on définit une balise <OPTION> pour chaque choix. SELECTED désigne un choix présélectionné. **Choix exclusif, sauf** option MULTIPLE.
<TEXTAREA COLS=num ROWS = num> libellé de la zone**</TEXTAREA>**	Zone de saisie d'un texte libre sur plusieurs lignes. COLS et ROW déterminent la taille de la zone.

B. Traitement des données d'un formulaire par PHP

Les informations saisies dans le formulaire sont transmises au serveur lors de l'appui sur le bouton de type SUBMIT. Voyons comment le programme PHP récupère ces informations, pour les traiter.

5.2 API mySQL en PHP

A.Eléments du site



B.Etapes du programme php

-ouvrir la connexion
-sélectionner la base
-envoyer la requête
-tester le résultat retourné
-si OK, traiter le résultat

C.Les fonctions de l'API

ANNEXE

Sessions

Lors d'une "même" visite sur un site, comportant plusieurs demandes successives de pages, le serveur traite chaque demande indépendamment, et n'établit aucun lien entre des demandes successives émanant d'un même client. La notion de "session" n'existe pas.

Plusieurs solutions existent pour résoudre ce problème :

1. **Variables d'environnement**.

   Le protocole HTTP comporte des variables systématiquement transmises au serveur par le navigateur. Par exemple, les variables REMOT_USER ou REMOTE_PASSWORD peuvent être définies lors d'accès à des sites sécurisés. Ces variables sont en nombre limité, fixées à l'avance, et dépendent du navigateur…

2. **Paramètres dans l'URL.**.

   Les informations sur la session peuvent être passées dans l'URL derrière le signe ?. Le programme peut les récupérer dans la variable QUERY_STRING. La taille de l'URL est limité, et ces information sont transmises en clair sur le réseau.

3. **Cookies**.

   Un *cookie* est une donnée que le navigateur conserve pour une période déterminée à la demande du serveur. Cette demande est effectuée dans l'en-tête d'une page HTML, avec une instruction SET_COOKIE. Le cookie est conservé dans un fichier, il ne disparaît pas après la fermeture du navigateur.

Déclaration d'une nouvelle base de données :

```
% mysql –u root –p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is __ to server version:_._._
Type 'help' for help

mysql> CREATE DATABASE Biblio
mysql> GRANT ALL PRIVILEGES ON Biblio.* TO adminBib@localhost
                              IDENTIFIED BY 'adminBib';
mysql> EXIT
```