Hiver 2021

Prise en mains de MySQL

Pour le cours de KBE, projet dirigé

Saliha Yacoub
CLG

Table des matières

Chapitre 1, Avant de commencer ! 3

Après installation, se connecter à la base de données en localhost: MySQL Workbench 4

Chapitre 2, Les types de données. **(Votre cours commence ici)** 6

Les entiers 6

NUMERIC et DECIMAL 6

FLOAT, DOUBLE et REAL 7

Les Dates 7

Pour les chaines de caractères : 7

Le type ENUM 7

Chapitre 3, Le modèle de données avec MySQL Workbench 9

Étape 0: Se connecter à votre BD distante: 9

Étape 1 : Créer la base de données 11

Étape 2, Créer le diagramme 11

Ajouter les relations entre les tables : cas de clé étrangère uniquement (cardinalités 1 :N d'un seul côté) 14

Ajouter les relations entre les tables : cas de clé étrangère qui devient clé primaire (cardinalités 1 : N des deux côté) 15

Étape 3 : enregistrez votre modèle 18

Étape 4, Générer le code SQL 18

Chapitre 4, Écrire des procédures stockées 20

Les procédures et les fonctions 20

Exemple 1, cas d'une fonction qui retourne un entier 20

Exemple 2, fonction avec paramètre 21

Exemple 3, cas d'une procedure stockée avec insertion 21

Exemple 4, procédure qui affiche une liste d'enregistrements 22

Exemple 5, procedure avec curseur 22

La structure de control: IF 24

Le IF simple 24

Le IF.. ELSE 24

Syntaxe générale: 24

Exemple 5, structure de control 24

Exemple 6, procédure pour vérifier qui vérifie l'existence d'un membre 25

Le triggers 25
Rappel 25
Syntaxe simplifiée 25
Exemple 1, ... 26
Exemple 2 26
Exemple 3 27
Chapitre 5, PDO par l'exemple 28
Introduction 28
Les classes importantes de PDO : 28
La classe PDO 28
La calsse PDOStatement : 28
La classe PDOException 28
Étapes pour le traitement des commandes SQL : 29
Étape 0 : Activer PDO : 29
Étape 1 : Connexion à la base de données : 29
Étape 3 : Envoyer les commandes SQL 31
Cas1 : *Requêtes exécutées une seule fois* 31
Cas2 : *Requêtes paramétrées* 35
Cas3 : *Appels de procédures stockées.* 38
Compléments PDO 42
PDOStatement::fetch 42
cursor_orientation 43
La classe PDOStatement 43
La classe PDO 44
Sources 45

Chapitre 1, Avant de commencer ! ……

Le cours qui suit se veut un aperçu du SGBD MySQL pour une prise en mains rapide de ce SGBD pour le développement de projets.

En aucun cas, il va dans les détails. Ce cours prend pour acquis que les étudiants ont les connaissances suivantes :

1. Connaissances avancées en SQL → ref : Oracle, session 2
2. De bonnes connaissances des notions de procédures stockées et de fonctions stockées→ ref : Transact-SQL, session 3
3. Connaissances des notions de triggers → ref : Transact-SQL, session 3
4. Connaissances de base à moyenne en normalisation →Session 2 et session3
5. Connaissances de base en conception de bases de données →session 3
6. Connaissances en PHP pour la partie PDO.

MySQL est le SGBDR libre et gratuit. MySQL est Open Source. Open Source (Standard Ouvert) signifie qu'il est possible à chacun d'utiliser et de modifier le logiciel. Tout le monde peut le télécharger sur Internet et l'utiliser sans payer aucun droit.

En général le SQL de MySQL est très semblable au SQL d'Oracle et de SQL Server que nous avons vu les dernières sessions.

Ce qui implique que TOUS ce que vous avez appris durant le cours de « Introduction aux bases de données » de la session 2 s'applique et reste valable pour les autres SGBDs à quelques exceptions près.

- La Commande CREATE TABLE reste la même. **MY SQL** a implémenté le concept d'incrémentation automatique de la clé primaire : **AUTO_INCREMENT**

- La commande ALTER Table est la même. De même que la commande DROP Table.

- La commande SELECT reste la même. Les jointures se font au niveau du FROM et non au niveau du WHERE.

- MY SQL tout comme Oracle et SQL Server sont des SGBD serveurs. (SQLite est un SGBD embarqué)

- Tous les SGBDs offrent une interface ou un logiciel de gestion des bases de données. Pour Oracle, nous l'avons vu, c'est SQL Developer. Pour MS SQL Server c'est SQL Server Management Studio, **pour MySQL c'est MySQL Workbench,** pour SQLite c'est SQLite DB Browser

- Les procédures stockées seront abordées plus loin.
- Vous pouvez télécharger le MySQL installer à http://dev.mysql.com/downloads/mysql/#downloads

Après l'installation de MySQL, un compte root est créé. Ce compte possède tous les droits. Il est donc important de créer des comptes avec accès restreint afin de garantir la sécurité des données.

Après installation, se connecter à la base de données en localhost: MySQL Workbench

Attention :

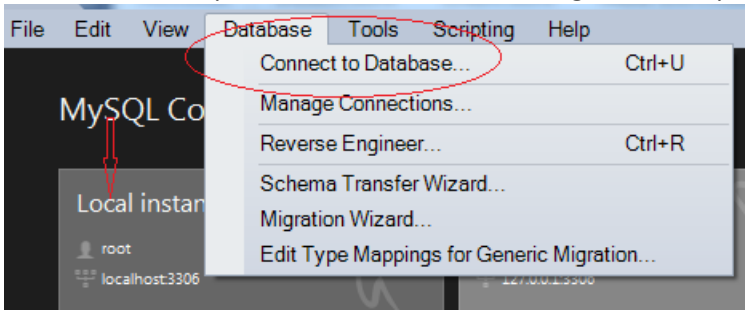
Ce qui suit, s'applique lorsque vous vous travaillez en local (localhost)

Ce ne sera pas notre cas….. mais c'est bon à savoir

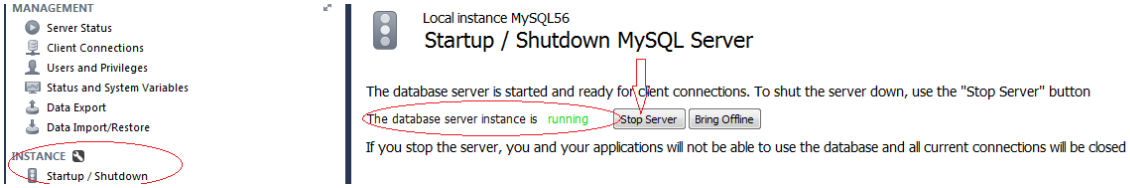
Étape 0 : Supprimer les comptes anonymes, utiliser le compte root pour le faire.

Étape 1 : On se logue avec le compte root et on crée notre user.

1. lancer MySQL Workbench, puis établir une connexion en tant que root comme suit :
   soit, par le menu Database, puis connect to Database,
   soit en double cliquant sur Local instance. L'usager root n'a pas de mot de passe.



2. vérifier que votre serveur est bien démarré. Sinon démarrez-le

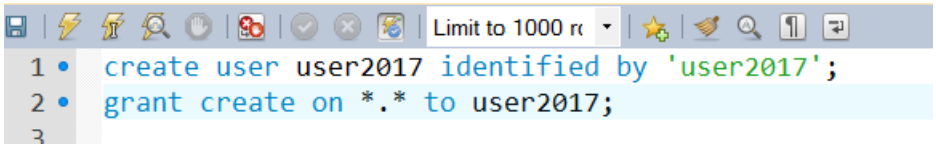


3. **Avec le compte root, créez votre propre usager**.
   Votre usager doit avoir les **8 premières lettres** de votre nom de famille. Sans espace et sans caractères spécial (/ »?&* etc..). Si votre nom de famille compte moins que 8 lettres compléter avec votre prenom (les premières lettres) de sorte que votre user ait 8 lettres.
   Donnez un mot de passe dont vous allez vous souvenir, idéalement assez complexe.
   Syntaxe et exemple.

```
create user user2017 identified by 'user2017';
```

4. **Toujours avec le compte root** : Pour que votre usager puisse utiliser les ressources du serveur MySQL, vous devez lui donner les droits nécessaires. On pourrait lui donner tous les droits sur tous les objets, mais par mesure de sécurité nous allons lui attribuer les droits au besoin.
   Le premier droit qu'on va accorder à votre user est celui de la création de la base de données sur tout le serveur. Ensuite, on val lui donner TOUS les droits sur ces propres objets.

```
grant create on *.* to user2017;
```



Étape 2 : On se logue avec notre user, et on créer la base de données.

1. On se connecte de la même façon que le root, cette fois en utilisant votre propre usager. On vérifie que tout fonctionne bien.
2. On crée la base de données qui va avoir le nom d'usager suivi de db1.
   Exemple

```
create database user2017db1;
```

Cette première base de données va nous servir pour nos exercices à venir.

Étape 3 : On se logue avec le root et on donne les droits sur la base de données.

1. Avec le compte root, donnez tous les droits à votre user sur votre base de données.

```
grant all on user2017db1.* to user2017;
```

Chapitre 2, Les types de données. **(Votre cours commence ici)**

http://dev.mysql.com/doc/refman/5.7/en/date-and-time-types.html

Les entiers

Type	Storage	Minimum Value	Maximum Value
	(Bytes)	(Signed/Unsigned)	(Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

Si vous voulez que les nombre soit positif (non signés) utilisé Unsigned
Exemple :

```
create table eleves  (num int  UNSIGNED, nom varchar(2));
```

NUMERIC et DECIMAL

NUMERIC et DECIMAL sont équivalents et acceptent deux paramètres : la précision et l'échelle.

- La précision définit le nombre de chiffres significatifs stockés, donc les 0 à gauche ne comptent pas. En effet 0024 est équivalent à 24. Il n'y a donc que deux chiffres significatifs dans 0024.
- L'échelle définit le nombre de chiffres après la virgule.

Dans un champ DECIMAL(5,3), on peut donc stocker des nombres de 5 chiffres significatifs maximum, dont 3 chiffres sont après la virgule. Par exemple : 12.354, -54.258, 89.2 ou -56.
DECIMAL(4) équivaut à écrire DECIMAL(4, 0).(Comme le NUMBER de Oracle)

FLOAT, DOUBLE et REAL

Le mot-clé FLOAT peut s'utiliser sans paramètre, auquel cas quatre octets sont utilisés pour stocker les valeurs de la colonne. Il est cependant possible de spécifier une précision et une échelle, de la même manière que pour DECIMAL et NUMERIC.

Quant à REAL et DOUBLE, ils ne supportent pas de paramètres. DOUBLE est normalement plus précis que REAL (stockage dans 8 octets contre stockage dans 4 octets), mais ce n'est pas le cas avec MySQL qui utilise 8 octets dans les deux cas. Je vous conseille donc d'utiliser DOUBLE pour éviter les surprises en cas de changement de SGBDR.

Les Dates

Vous pouvez spécifier les valeurs des colonnes DATETIME, DATE

Le type DATE est prévu lorsque vous souhaitez stocker une date. MySQL affiche les valeurs de type DATE au format 'AAAA-MM-JJ'. L'intervalle de validité va de '1000-01-01' à '9999-12-31'.

Le type DATETIME est prévu lorsque vous souhaitez stocker une date et une heure. MySQL affiche les valeurs de type DATETIME au format 'AAAA-MM-JJ HH:MM:SS'.

Pour les chaines de caractères :

Pour stocker un texte relativement court (moins de 255 caractères), vous pouvez utiliser les types CHAR et VARCHAR. Ces deux types s'utilisent avec un paramètre qui précise la taille que peut prendre votre texte (entre 1 et 255).

La différence entre CHAR et VARCHAR est la manière dont ils sont stockés en mémoire. Un CHAR(x) stockera toujours *x* caractères, en remplissant si nécessaire le texte avec des espaces vides pour le compléter, tandis qu'un VARCHAR(x) stockera jusqu'à *x* caractères (entre 0 et *x*), et stockera en plus en mémoire la taille du texte stocké.
Si vous entrez un texte plus long que la taille maximale définie pour le champ, celui-ci sera tronqué.

Pour stocker un text plus que 255 caractères, utilisez le type TEXT

Le type BLOB est également utilisé pour les gros fichiers

Le type ENUM

C'est un string avec un ensemble de valeurs. (il peut remplacer la contrainte CHECK dans certains cas)

Exemple1 :

```
create table prog(id enum ('inf','tge','tad'),
constraint pkpro primary key(id),
nom varchar(40));
```

Exemple 2:

```
CREATE TABLE PROGRAMMES
(
 CODEPRG  int unsigned,
 CONSTRAINT PKCODEPRG PRIMARY KEY(CODEPRG),
 NOMPROG VARCHAR(30)
);
```

Remarque

1. Pour donner un nom à la contrainte de clé primaire, il faut qu'elle soit au niveau table.

```
CREATE TABLE ETUDIANTS
(
 NUMAD int unsigned auto_increment,
 CONSTRAINT PKNUMAD PRIMARY KEY(numad),
 NOM VARCHAR(20) NOT NULL,
 PRENOM VARCHAR(20),
 CODEPRG int unsigned,
 VILLE VARCHAR(30),
 TELEPHONE VARCHAR(20),
  CONSTRAINT FKCODEPRG FOREIGN KEY(CODEPRG) REFERENCES
PROGRAMMES(CODEPRG)
);
```

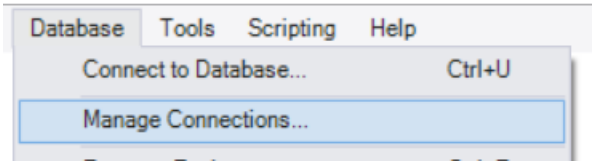
Chapitre 3, Le modèle de données avec MySQL Workbench

Étape 0: Se connecter à votre BD distante:

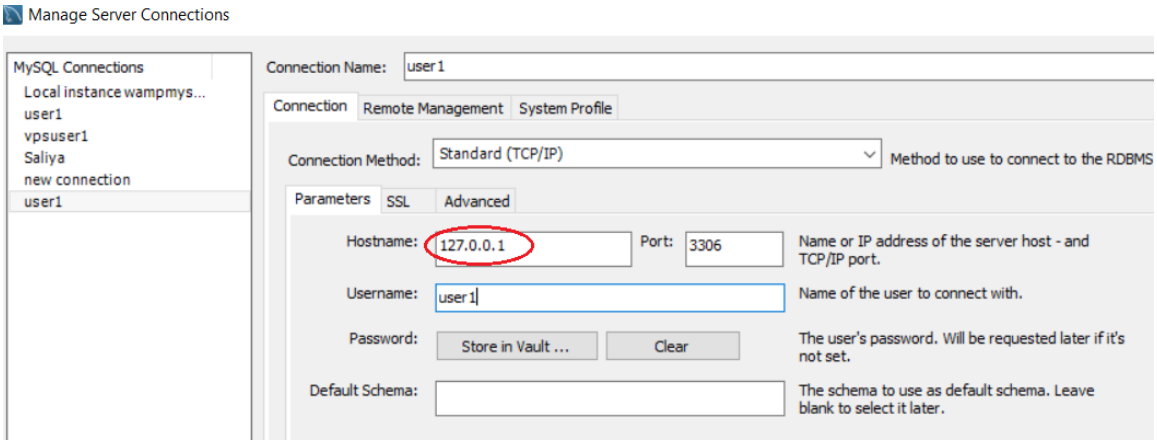
Attention :

Ce qui suit, est ce que nous allons utiliser

1. Par le menu Database, choisir Manage Connections



2. Puis choisir New
3. Renter les informations de connections



4. Puis **tester**. Le système vous demande de rentrer un mot de passe.

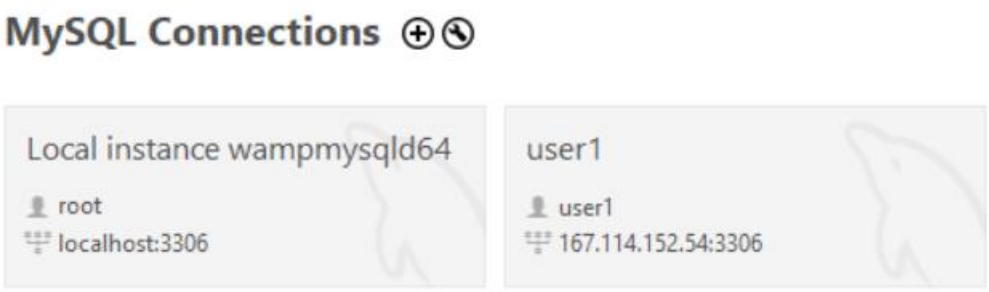
Attention :

Rentrer la bonne adresse IP. Celle donnée par les profs

Attention :

NE PAS ENREGISTRER LE MOT DE PASSE DANS LE VAULT

5. Des avertissements de sécurité von s'afficher mais continuer
6. Fermer MySQL Workbench
7. Lancer à nouveau MySQL Workbench. Votre connexion s'affiche.

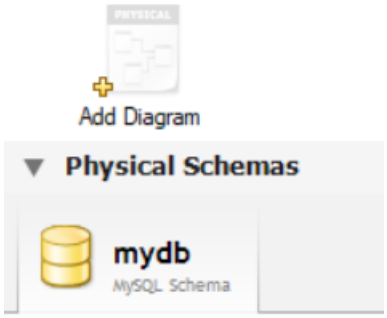


8. Double cliquez sur votre connexion
9. Fournir votre mot de passe.
10. Vous avez une des meilleures interfaces d'accès au SGBD. Meilleure que SQL Developer, meilleure que SQL Server Management Studio
11. Vous pouvez commencer à créer vos tables. Le SQL est standard.

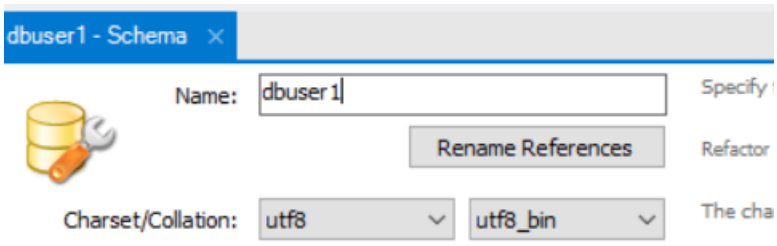
Étape 1 : Créer la base de données.

Dans votre cas la BD est déjà créée, car vous n'avez pas de droits sur le serveur. Par contre il faudra indiquer le bon nom de la BD pour le système. Au moment de l'exécution du script SQL, la commande USE va pointer sur votre BD.

Fichier → nouveau modèle



Double cliquer sur mydb, puis donner le nom de **votre base de données. Cette étape est importante pour la génération du code SQL**



Étape 2, Créer le diagramme

Cliquez sur Add Diagram. Une interface conviviale pour ajouter votre modèle de données s'affiche. Repérez l'outils table, et l'ensemble des relations.

Lorsque vous créez votre modèle relationnel, en principe vous pensez à déterminer les éléments suivants :

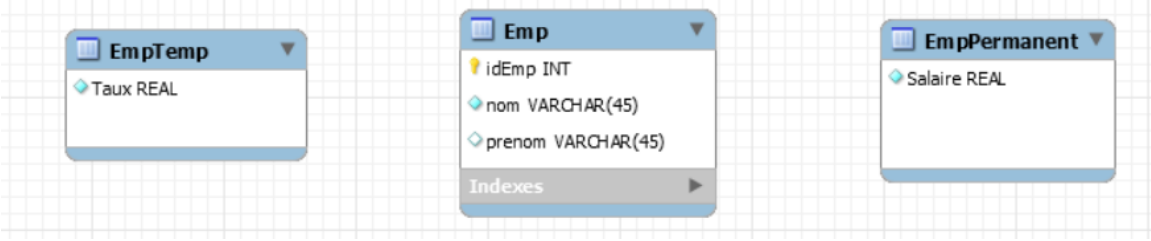
1. Vos tables et leurs colonnes ou attributs
2. Pour chaque colonne, on détermine :
    a. Le type des données : INT, Varchar(n),…
    b. Le type de contrainte, PK, CK not null…
    c. Le nom des contraintes
3. Le lien entre les différentes tables. Ce lien est déterminé par vos règles de gestion. Ce lien représente la contrainte de clé étrangère (FK). Pour l'attribut de la FK, il faut déterminer s'il est **Optionnel ou Obligatoire** et s'il est **Unique ou Multiple**. Lorsque vous

faîtes un modèle relationnel, les liens entres les tables nous renseignent sur ces contraintes.

MySQL Workbench, pour les modèles de données ne fonctionne pas comme MS SQL Server. Il fonctionne un peu comme DataModeler.

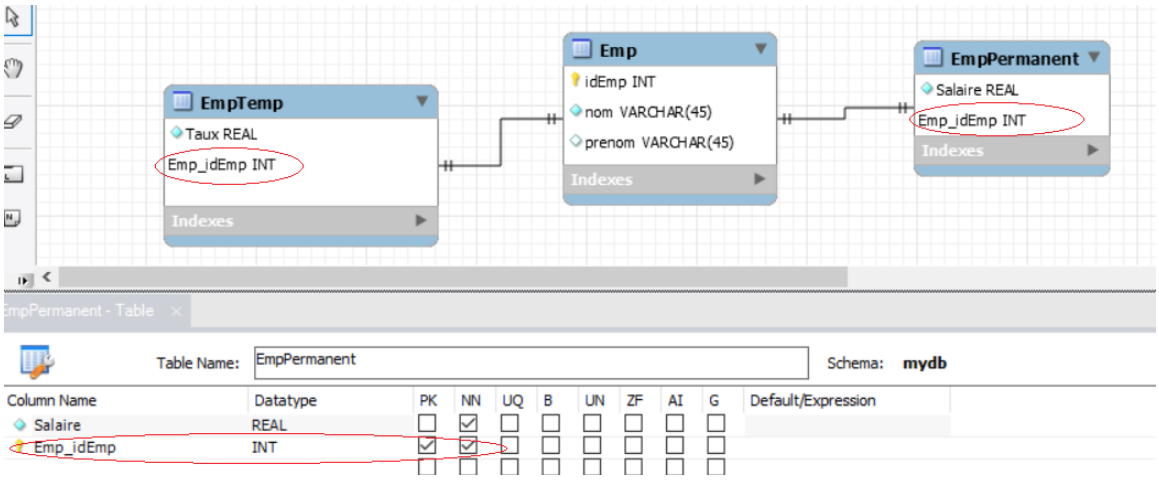
Outils		Significations
		Permet de créer une table.
1		La relation 1 :1 (pointillé) indique l'optionalité et l'unicité de la FK dans la table où la PK a migré. Comme par exemple : Un étudiant a 0 ou 1 stage. Un stage concerne 0 ou un étudiant. Lorsque la clé primaire migre d'une table A pour devenir une FK dans la table B, la FK n'est pas une clé primaire
2		La relation 1 :1 Ligne pleine indique l'obligation et l'unicité de la FK dans la table où la PK a migré. Comme par exemple : un employé a un et un contrat, un contrat concerne un et un seul employé. Ici, il n'y a pas d'employé sans contrat, et il n'y a pas de contrat sans employé. La clé étrangère est également une clé primaire.
3		La relation avec la fourche en pointillé: indique l'optionnalité et la multiplicité de la FK. Lorsqu'on veut avoir uniquement une FK (pas de PK)
4		La relation avec la fourche en trait plein: indique l'obligation et la multiplicité de la FK. Utilisée lorsqu'on souhaite avoir une FK et PK en même temps (Clé primaire composée)
5		La relation avec les fourches des deux côtés, indique la multiplicité et l'obligation. Elle génère des tables avec des FK et des clé primaires composées. C'est le cas : un étudiant est inscrit dans plusieurs cours et un cours a plusieurs programmes.
6		Utilisé lorsqu'il existe déjà un attribut destiné à être une FK. (un peu ce que vous avez l'habitude de faire

Pour la relation numéro 3, on examine le modèle initial suivant :



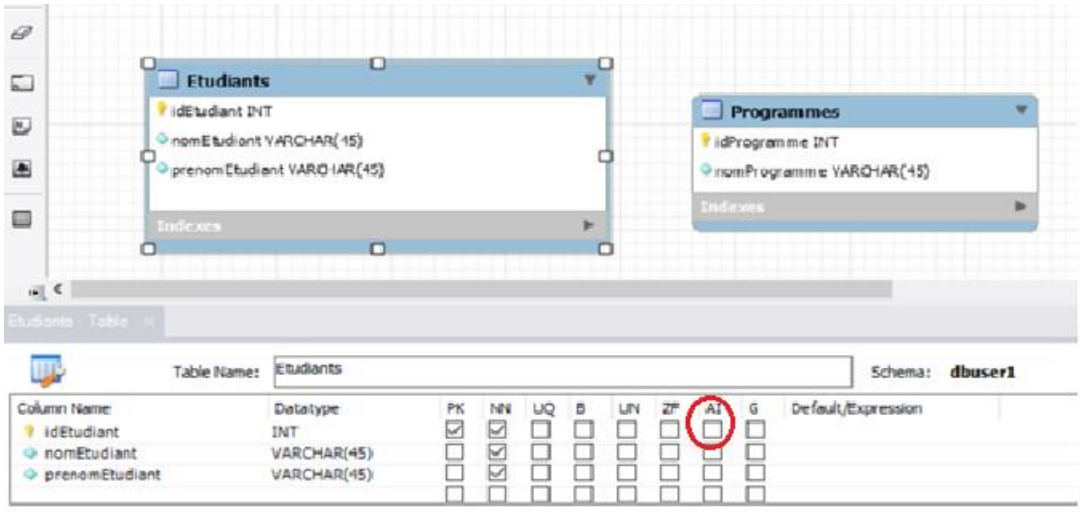
Remarquez que les deux tables EmpTemp et EmpPremanet n'ont aucune clé primaire .

Après application de la relation 3 (1-1) on obtient ceci :



Remarquez pour l'attribut idEmp :

1. La clé primaire idEmp de la table EMP a migré dans la table EmpTemp et EmpPermanent pour devenir une clé étrangère dans chacune des tables.
2. La clé primaire des tables EmpTemp et empPermanent est la clé la clé étrangère idEmp.
3. Évidemment, comme idEmp est clé primaire dans les tables EmpTemp et EmpTemporaire alors elle est obligatoire.
4. Vous pouvez décidez des actions à entreprendre en cas de suppression d'un enregistrement de la table Emp lié à un enregistrement de la table EmpTemp ou EmpPermanent.
5. Dans la plupart des cas, le ON DELETE ça va être NO ACTION, mais vous pouvez prendre CASCADE si c'est ça qu'on veut.



1. Il faut donner un nom à votre table
2. Ajouter des colonnes
3. Pour chaque colonne, définir le type de données et les contraintes
4. Si vous voulez que votre clé soit Auto_Increment, cochez la case en rouge.
5. Ajouter les relations entre les tables.

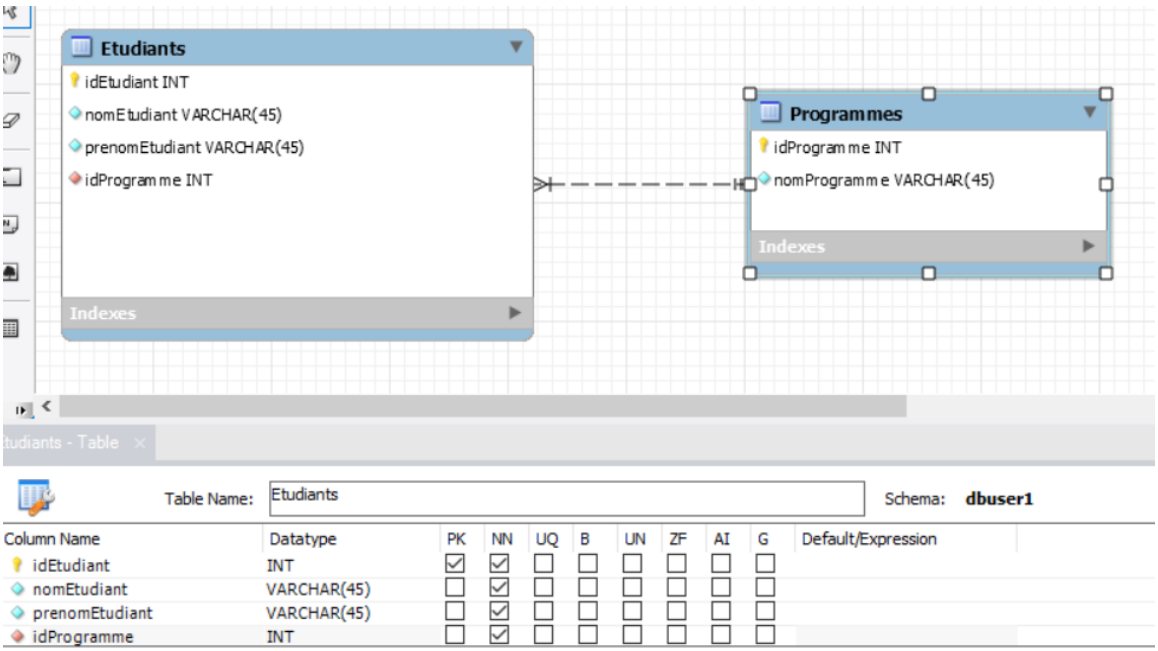
Ajouter les relations entre les tables : cas de clé étrangère uniquement (cardinalités 1 :N d'un seul côté)

Remarque : Il existe deux façons de créer les liens entre les tabes :

Façon numéro 1 : Aucun attribut de clé étrangère n'est encore créé. (La façon suggérée)

Ces deux tables n'ont pas encore de lien. Mais on souhaite mettre une FK dans la table Etudiants et un lien entre ces deux tables. Puisque pour l'instant il n'y a pas d'attribut de FK dans la table Etudiant, **on va le créer à l'aide de la relation 3 du tableau** : **une fourche avec des pointillées**. On procède comme suit :

a. On sélectionne la relation
b. On clique d'abord sur la table Etudiants
c. On clique ensuite sur la table programmes.
d. Vous pouvez renommer la colonne de la FK
e. Vous devez quand même vérifier si c'est ce qui est attendu.



2- Façon numéro 2 : Il y a un attribut qui va être une FK.

Dans le modèle suivant, le idProgramme dans la table COURS pourrait être une FK.

Pour créer le lien entre les deux tables Programmes et cours, on utilise le lien numéro 6 dans le tableau précedent.

a. Sélectionne le lien
b. On clique sur idProgramme dans Cours
c. On clique sur IdProgramme dans Programmes
d. On vérifie.

Ajouter les relations entre les tables : cas de clé étrangère qui devient clé primaire (cardinalités 1 : N des deux côté)

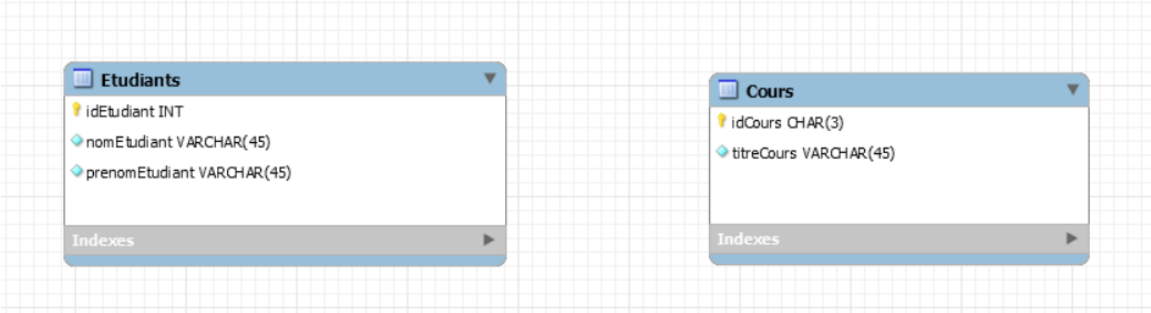
Pour les relations de plusieurs à plusieurs, il existe aussi deux façons de faire : soit la tables contenant les attributs de clé étrangères est créé soit, cette table va se créer à même la relation.

Façon no1 : la table de lien n'est pas créée (La façons suggérée)

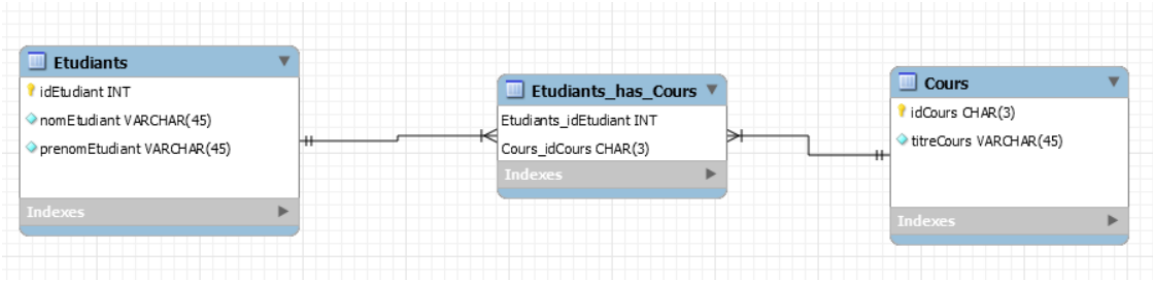
On veut créer la table Resultat entre Etudiants et cours. La table qui sera créée aura

1. Un attribut idEtudiant, FK (ref Etudiants)
2. Un attribut idCours FK(ref Cours)
3. Un attribut note
4. (idEtudiant,idCours)PK.

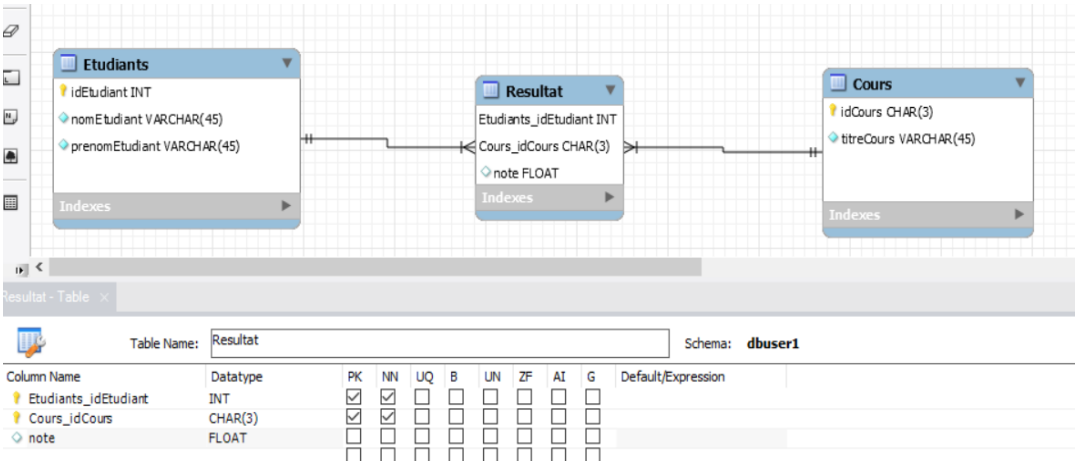
Rien de plus simple :



1. On sélectionne la relation ayant une fourche de chaque côté. Relation 5 du tableau
2. On clique sur idEtudiant, puis
3. On clique sur idCours
4. On obtient le diagramme la table suivante :Etudiant_has_cours

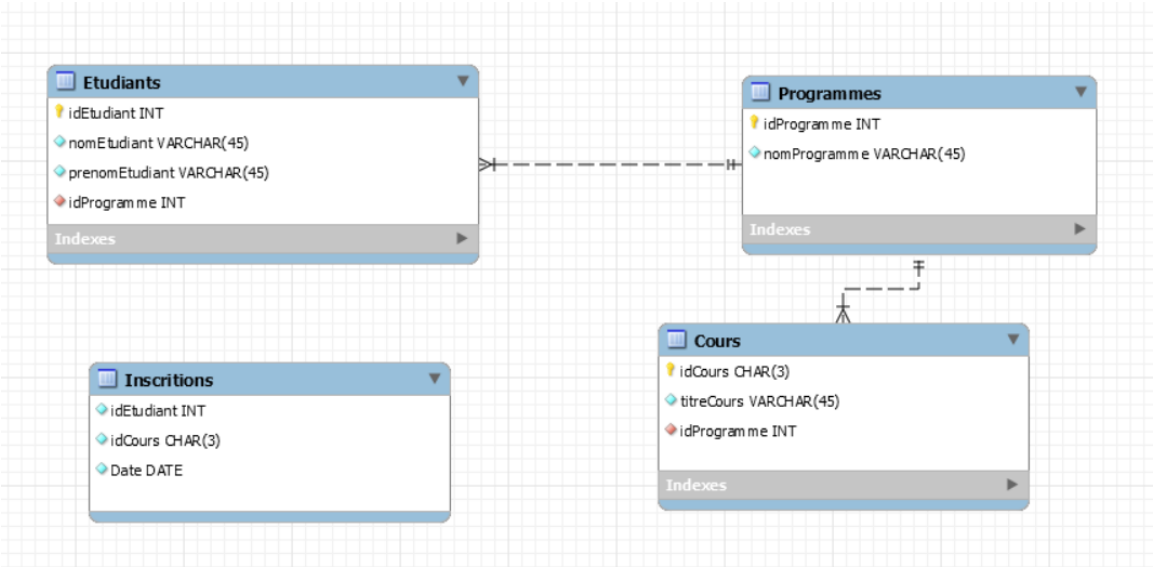


5. Renommer la table pour Resultats
6. Ajouter l'attribut note
7. Vous obtiendrez ceci



Façon no2 : Voici le modèle suivant : La table inscription est créée avec les attributs qui vont être des FK mais aucune relation n'est définie. On veut définir :

- la relation entre etudiants et inscription
- la relation entre cours et inscription
- les FK et la PK

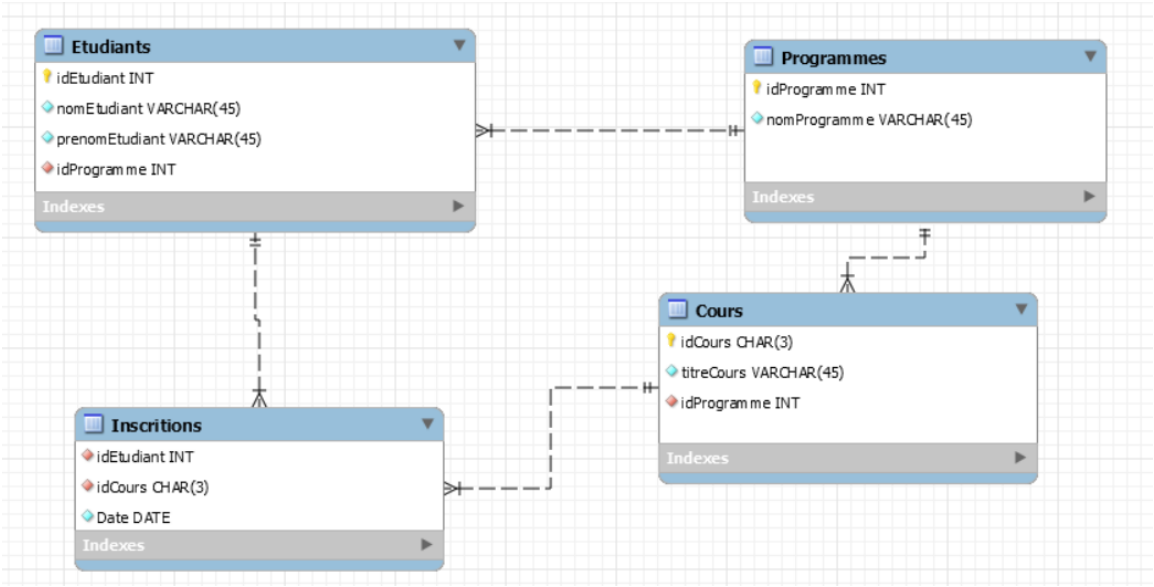


On procède comme suit :

Pour créer le lien entre les deux tables Programmes et cours, on utilise le lien encerclé en rouge du tableau précédent.

1. Sélectionne le lien numéro 6 dans le tableeau
2. On clique sur idEtudiants dans INSCRIPTION , puis
3. On clique sur IdEtudiants dans Etudiants

4. On clique sur IdCours dans INSCRIPTIONS, puis
5. On clique sur idCours dans cours.
6. On obtient le diagramme suivant

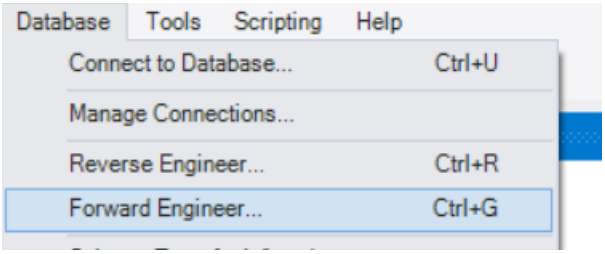


7. On double clique sur la table INSCRIPTIONS
8. Et on définit idEtudiant comme PK, on définit idCours comme PK
9. Les liens en pointillé sur la table inscription deviennent des lien en traits plein.

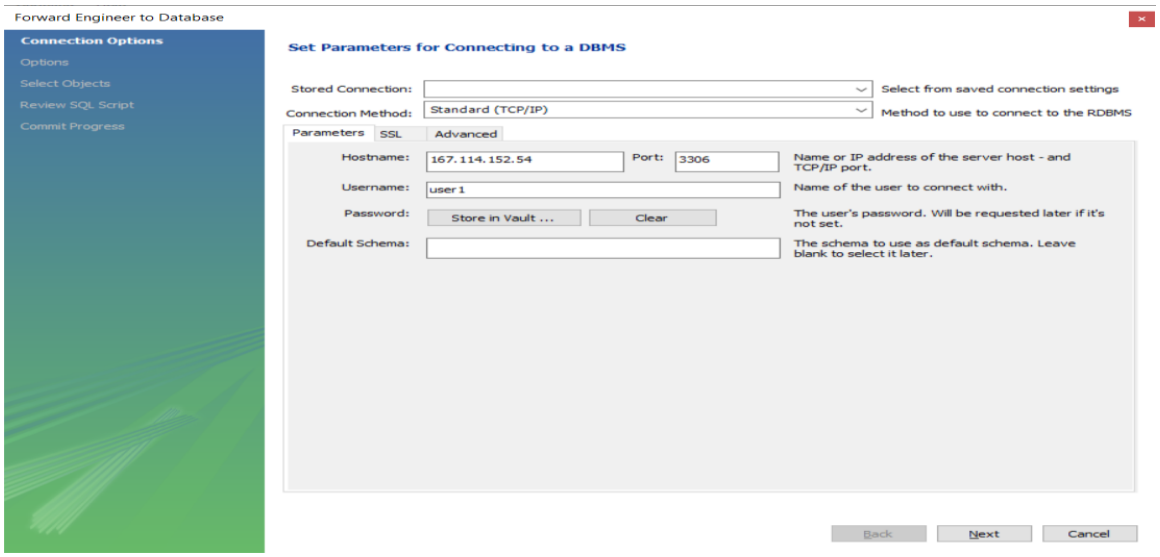
Étape 3 : enregistrez votre modèle

Étape 4, Générer le code SQL

1. Par le menu Database, choisir Forward Engineer



2. Vérifier que ce sont vos paramètres de connexion



3. Faire suivant, puis suivant
4. Quand vous aurez cette fenêtre, enlever le mot VISIBLE. (juste VISIBLE, laisser la virgule)

5. Vérifier votre code. Enregistrer puis suivant. Puis close.
6. Vérifier que vos tables sont bel et bien créées

Chapitre 4, Écrire des procédures stockées

Les procédures et les fonctions

À quelques exceptions, le corps des procédures et fonctions sont identiques à ce que nous avons vu avec MS SQL Server

Certaines différences sont à considérer.

1. Il faut un **delimiter** spécial dans la procédure, fonction ou trigger.
2. Il faut utiliser SELECT ……..INTO lorsque la fonction retourne une valeur (une seul).
3. Il faut le mot réservé DECLARE pour la déclaration des variables
4. Le type de paramètre (IN ou OUT) est transmis en premier.
5. Pour une fonction, puisque le type (IN OUT) des paramètres est toujours en IN, alors pas besoin de le préciser.
6. Remarquez le S du returnS
7. Pour les appels de fonctions, on utilise SELECT
8. Pour les appels de procédures on utilise CALL

Exemple 1, cas d'une fonction qui retourne un entier

```
delimiter $$

create function calculer() returns integer

begin

declare total integer;

select count(*) into total from Etudiants;

return total;

end $$
```

L'appel de la fonction se fait par un simple SELECT

```
select calculer() ;
```

Exemple 2, fonction avec paramètre.

```
delimiter |
create function compterEtudiantProgramme(pcodep int) returns integer
begin
declare total integer;
select count(*) into total from Etudiants
where idProgramme =pcodep;
return total;
end |
```

L'appel de la fonction se fait :

```
select compterEtudiantProgramme(420);
```

Exemple 3, cas d'une procedure stockée avec insertion

```
delimiter |
CREATE  PROCEDURE insertEtudiant
(
in pidetudiant int,
in pnom varchar(20),
in pprenom varchar(20),
in pcode int
)
BEGIN
insert into Etudiants values(pidetudiant,pnom,pprenom,pcode);
commit;
END|
```

L'appel de la procédure se fait :

```
call insertEtudiant(22,'Lenom','Leprenom',420);
```

Exemple 4, procédure qui affiche une liste d'enregistrements

```
delimiter |

create procedure afficherEtudiantProgramme(in pnomProgramme varchar(30))

begin

select nomEtudiant, prenomEtudiant from (Etudiants e inner Join Programmes p
on e.idProgramme= p.idProgramme)

where nomProgramme =pnomProgramme;

end |
```

l'appel se fait comme suit

```
call afficherEtudiantProgramme('informatique');
```

Exemple 5, procedure avec curseur

```
DELIMITER |
CREATE PROCEDURE majCommade()
begin
DECLARE varprix integer ;
 DECLARE varidArticle integer;
 DECLARE lafin INT DEFAULT 0;
 DECLARE prix_cursor CURSOR FOR SELECT idArticle,prix FROM Articles;
 DECLARE CONTINUE HANDLER FOR NOT FOUND SET lafin = 1;
 OPEN prix_cursor ;
     majmontant: LOOP
      FETCH prix_cursor INTO varidArticle,varprix;
                  IF lafin= 1 THEN
                        LEAVE majmontant;
                  END IF;
      update Ligne_commande set Montant =quantiteCommande*varprix
               where idArticle =varidArticle;
      end loop majmontant ;
CLOSE prix_cursor;
END|
```

Dans l'exemple précèdent, on met à jour la colonne Montant de la table Ligne_Commande comme suit :

Le montant est égal à la quantité qui est dans la table Ligne_Commande * le pris qui est dans la table Articles.

Contenu de la table Articles :

idArticle	descriptions	prix
1	Imprimantes Laser HP 8000	700
2	Écrans tactiles 19p	550
3	Routeurs sans fils Azus	200
4	Disques durs SSD 500 Go	175
NULL	NULL	NULL

Contenu de la table Ligne_commande après la mise à jour

idcommande	idArticle	quantiteCommande	montant
10	1	2	1400
10	2	3	1650
10	3	5	1000
10	4	6	1050
11	4	2	350
11	3	3	600
12	3	5	1000
13	1	1	700

mmande 1

Pour mettre à jour le montant, nous avons besoin de chercher le prix de chaque article. Nous avons donc besoin d'un curseur.

Le fonctionnement des curseurs est le même pour tous les SGBD. Le déclarer explicitement avec la requête SELECT, l'ouvrir (OPEN), lire son contenu avec FETCH. Il faut utiliser un LOOP pour le balayer en entier.

Dans notre cas, au fur et à mesure qu'on lit le curseur, on fait les mises à jour dans la table Ligne_commande.

Tout curseur ouvert, doit être fermé.

Comment indiquer la fin de la lecture ? En utilisant une variable qui indique si nous sommes arrivés à la fin du curseur.

La structure de control: IF

Le IF simple

```
IF condition THEN
  statements;
END IF;
```

Le IF.. ELSE

```
IF condition THEN
  statements;
ELSE
  else-statements;
END IF;
```

Syntaxe générale:

```
IF condition THEN
  statements;
ELSEIF elseif-condition THEN
  elseif-statements;
...
ELSE
  else-statements;
END IF
```

Exemple 5, structure de control

La fonction qui suit vérifie si un nom existe dans la table Etudiants. Si oui, elle retourne 1, sinon elle retourne 0

```
delimiter |

create  function verifierNom(palias varchar(30)) returns integer

begin

Declare reponse integer;

        if palias in (select nomEtudiant  from Etudiants where nomEtudiant =palias)

        then set reponse=1;

        else  set reponse=0;

        end if;

return reponse ;
```

```
end |
```

Exemple 6, procédure pour vérifier qui vérifie l'existence d'un membre

Vérifier que le user et le mot de passe sont corrects avec une procédure. L'opérateur **EXISTS** s'utilise comme le IN dans les sous requêtes. S'il est utilisé dans le SELECT et non dans le WHERE, alors il retourne 1 ou 0 selon qu'il existe des enregistrements ou non

```
delimiter |

CREATE  PROCEDURE verifierMembreExiste(in pseudo varchar(20),in pmpasse varchar(20))

BEGIN

declare resultat int;

declare reponse char(1);

select exists(select alias,mpass from membres where alias = pseudo and mpass=pmpasse) into
resultat;

        if resultat=1  then set reponse='Y' ;

        else set  reponse= 'N';

        end if;

select reponse;

END |
```

Le triggers

Rappel

Les triggers sont des procédures stockées qui s'exécutent automatiquement quand un événement se produit. En général cet événement représente une opération DML (Data Manipulation Language) sur une table. Les instructions DML doivent inclure INSERT, UPDATE ou DELETE. Les triggers de MySQL sont très différents des triggers MS SQL SERVER.

Syntaxe simplifiée

```
CREATE  TRIGGER nomtrigger
{BEFORE | AFTER }
{INSERT | UPDATE | DELETE}
ON Nomdetable FOR EACH ROW
BLOC SQL
```

Lors de la création d'un trigger il faut savoir que :

1. Il faut commencer par un DELIMITER
2. Une seule opération DML par Trigger.
3. Lors d'un INSERT la nouvelle valeur de la colonne est NEW.nomColonne →INSERTED
4. Lors d'un DELETE l'ancienne valeur est OLD.nomColonne -→DELETED
5. Lors d'un update l'ancienne valeur est OLD.nomColonne et la nouvelle valeur est NEW.nomcolonne.
6. FOR EACH ROW indique que le trigger va s'exécuter pour chaque ligne de la table
7. Il existe une fonction SIGNAL qui permet d'arrêter l'opération DML en cours si les conditions ne sont vérifiées.

Exemple 1,

```
DELIMITER |;

 CREATE TRIGGER suppresion

before  delete ON DEPARTEMENTS

for each row

BEGIN

update employes set codedep = null where codedep=old.codedep;

END |;
```

Dans l'exemple 1, on décide de mettre à NULL les départements qui seront supprimés dans la table employes avant de supprimer dans la table employes.

Si on voulait faire une suppression en CASCADE, on aurait utilisé un DELETE au lieu du UDATE

```
DELETE FROM EMPLOYES WHERE  EMPLOYES.CODEdep = OLD.codedep;
```

Exemple 2

```
delimiter |

create trigger ctrSalaire

before update on employes

for each row

begin

if(new.salaire < old.salaire) then

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le salaire est bas';

end if;
```

```
end |
```

Exemple 3

```
DELIMITER |;
create trigger empsal before insert on employes
for each row
begin
declare sal decimal;
select avg(salaire) into sal from employes
where codedep =new.codedep;
if new.salaire is null
then set new.salaire =sal;
end if;
end |;
```

Chapitre 5, PDO par l'exemple

Introduction

Il existe 3 façons d'accéder aux bases de données par PHP :

Mysql qui utilise des requêtes mais pas de procédures stockées,

Mysqli très performant puisque du côté serveur; elle est propre à MYSQL(un peu comme OCI de Oracle)

PDO qui est du côté PHP (peut-être moins performant mais très standard puis que cette méthode est du côté application (un peu comme le JDBC)

PDO est donc une API d'accès aux données (n'importe que SGBD), orienté objet utilisé par PHP. Il est disponible avec les versions PHP 5 et plus.

PDO présente plusieurs avantages (mis à part qu'il est objet) il s'utilise avec les procédures stockées et les requêtes paramétrées ce qui permet de se protéger contre les injections SQL.

Les classes importantes de PDO :

La classe PDO :

Une instance de PDO représente la connexion à une base de données. Le plus souvent une seule instance de PDO par exécution de PHP.
Cette classe contient entre autre les méthodes exec(), query() et prepare()

exec() : permet d'exécuter une requête DML
query() : permet d'exécuter une requête SELECT
prepare() : utilisée pour une requête avec paramètres ou procédure stockées

La calsse PDOStatement :

Une instance de PDOStatement représente une requête vers la base de données. Permet de préparer la requête puis de consulter son résultat.
Cette classe contient entre autre les méthodes fetch(), bindParmeter(), rowCount(),execute()

fetch() : utilisée pour lire un enregistrement courant. Une ligne la fois. (curseur)

bindParameter() : pour lier les paramètres

execute() : exécuter une requête.

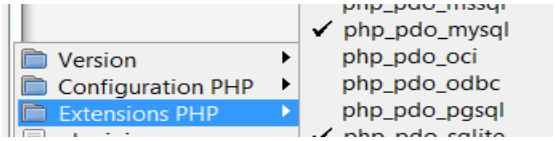
La classe PDOException

Pour la gestion des Exceptions.

Étapes pour le traitement des commandes SQL :

Étape 0 : Activer PDO :

Normalement, PDO est activé par défaut. Pour le vérifier (voir la figure suivante), faites un clic gauche sur l'icône de WAMP dans la barre des tâches, puis allez dans le menu PHP / Extensions PHP et vérifiez que php_pdo_mysql est bien coché.



Étape 1 : Connexion à la base de données :

Comme tout SGBD, pour se connecter à la base de données, il faut avoir les paramètres suivants :

- Nom du serveur ou son adresse IP (ou nom de l'hôte). C'est l'adresse de l'ordinateur où MySQL est installé.
- Nom de la base de données sur laquelle vous voulez obtenir une connexion
- Nom du user : nom de l'usager qui se connecte à la base de données.
- Mot de passe du user : mot de passe de l'usager qui se connecte :

Toute connexion ouverte doit être fermée

Syntaxe pour ouvrir une connexion :

```
$connexion = new PDO('mysql:host=$hote;dbname=$basededonnee;charset=utf8', $user, $mpasse);
```

Exemple :

```php
<?php
try
    {
    $mybd = new PDO('mysql:host=167.114.152.54; dbname = dbuser1; charset=utf8', 'user1', 'user1');
    echo('connexion reussie');
   }

   catch (PDOException $e)
    {
       echo('Erreur de connexion: ' . $e->getMessage());

       exit();

    }

$mybd=null;
?>
```

Pour fermer la connexion :

```
$mybd=null;
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>connexion  </title>
        <meta charset="utf-8" />
    </head>
    <body>
        <h2>Tester une connexion</h2>
  <?php
try
      {
      $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');
    echo('connexion reussie');
      }
    catch (PDOException $e)
      {
       echo('Erreur de connexion: ' . $e->getMessage());
       exit();
      }
  $mybd=null;
?>
    </body>
</html>
```

Certains sites vous proposerons la fonction die() qui est équivalente à exit();

```
catch (Exception $e)
{ die('Erreur : ' . $e->getMessage());}
```

Étape 3 : Envoyer les commandes SQL

Cas1 : *Requêtes exécutées une seule fois*

Attention : 

Cette façon de faire n'Est pas recommandée

La méthode **PDO ::exec() :** Cette méthode permet d'envoyer une requête de type DML à la base de données. Elle retourne un entier indiquant le nombre de lignes affectées

public int PDO::exec (string $statement)

```
 <?php
try
      {
       $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');
   echo('connexion reussie') . '<br />';
   $insertion = $mybd->exec("INSERT INTO Programmes
(idProgramme,nomProgramme) VALUES (520,'Un autre')");
    echo('total insertion est ' . $insertion) ;
      }
   catch (PDOException $e)
      {
       echo('Erreur de connexion: ' . $e->getMessage());
        exit();
      }
  $mybd=null;
?>
```

La méthode **PDO ::query()** Cette méthode permet d'envoyer une requête SELECT sans paramètres à la base de données. Elle retourne un PDOStatement (ou un curseur)

```
public PDOStatement PDO::query ( string $statement )
```

Pour lire le curseur on utilise la méthode **fetch()** (méthode sql ou PL/SQL ou transact-SQL) et évidement une boucle while. Exemple :

```
while ($donnees = $curseur->fetch())
{        affichage        }
```

La commande FETCH permet de lire ligne par ligne le contenu du curseur. À chaque fois que cette commande est appelée, le curseur avance au prochain enregistrement dans l'ensemble actif. Le résultat de la lecture est envoyé la table de nom donnees

Dans l'exemple qui suit le fetch() est par défaut. La lecture du tableau de données se fait par l'indice de colonne. L'indice de la première colonne est **ZÉRO. (comme en C#)**

Parfois, il est nécessaire de dire à votre programme comment vous voulez lire vos données. Comme par exemple accéder au tableau par le nom des colonnes.

PDO::FETCH_ASSOC: retourne un tableau indexé par le nom des colonne;

PDO::FETCH_OBJ: retourne un objet dont les propriétés correspondent aux nom de colonnes.

Exemple 1, le fetch() par défaut

```
  <?php
try
      {
       $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');
      echo('connexion reussie') . '<br />';
      echo ('affichage du contenu de la table programmes') . '<br />';
       $resultat = $mybd->query("SELECT * FROM Programmes ");
    while ($donnees = $resultat->fetch())
            {
             echo $donnees[0] . $donnees[1] .'<br />';
            }
      echo ($resultat->rowCount()); // nombre totale de lignes
      $resultat->closeCursor();// fermer le curseur
      }
   catch (PDOException $e)
       {
        echo('Erreur de connexion: ' . $e->getMessage());
        exit();
       }
$mybd=null;
?>
```

Exemple2 : fetch() associatif, lecture par les noms de colonnes

```
  <?php
try
      {
       $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');


      echo('connexion reussie') . '<br />';
      echo ('affichage du contenu de la table programmes') . '<br />';
      $resultat = $mybd->query("SELECT * FROM Programmes ");
      $resultat->setFetchMode(PDO::FETCH_ASSOC);
       while ($donnees = $resultat->fetch())
            {
echo $donnees['idProgramme']. $donnees['nomProgramme'] . '<br />';
            }
echo ($resultat->rowCount()); // nombre totale de lignes
$resultat->closeCursor();// fermer le curseur
      }
   catch (PDOException $e)
        {
       echo('Erreur de connexion: ' . $e->getMessage());
        exit();
        }
  $mybd=null;
?>
```

Cas2 : *Requêtes paramétrées*

Attention : 

Recommandée : Bonne pratique quand vous n'avez pas de procédures stockées.
Pour les requêtes DML ça peut être très intéressant. Pour les requêtes SELECT… personne ne veux voir votre requête dans le code PHP

Les requêtes paramétrées sont incontournables lorsque nous essayons d'accéder à une base de données via le web. Elles ont un avantage majeur qui est de réduire considérablement les injections SQL. De plus ce sont des requêtes préparées donc exécutées plus d'une fois et précompilées.

Le paramètre est représenté par un ? ou par :nomParametre

Et qu'allons-nous utiliser pour les requêtes paramétrées? Le PDOStatement

Le fonctionnement de l'exécution de requêtes paramétrées est :

La methode **prepare() :** cette méthode de la classe PDO permet de passer une requête paramétrée au SGBD.

public PDOStatement **PDO::prepare** (string $statement)

La methode **BindParam()** de la classe PDOStatement permet de lier les variables aux valeurs.

public bool **PDOStatement::bindParam** ($parameter , &$variable [, int $data_type = PDO::PARAM_STR [, int $length]])

Exemples:

$stmt1->bindParam(1, $nom);

$stm->bindParam(2, $race, PDO::PARAM_STR);

$stm->bindParam(2, $race, PDO::PARAM_STR,20);

La méthode **execute()** permet d'exécuter la requête avec ses paramètres. Elle retourne true ou false selon le succès de l'exécution de la requête

public bool **PDOStatement::execute** ([array `$input_parameters`])

Exemple de requête INSERT

```
  <?php
try
{
 $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');
      echo('connexion reussie') . '<br />';
      echo ('insertion avec paramètres') . '<br />';
  $stmt1 = $mybd->prepare("INSERT  INTO Programmes (idProgramme,
nomProgramme) VALUES  (?, ?)");
// lier les paramètres
$stmt1->bindParam(1, $id);
$stmt1->bindParam(2, $nom);
// affectation de valeurs aux paramètres
$id = 12;
$nom = 'SNF';
// executer la requête
$total= $stmt1->execute();
echo('total insertion est ' . $total);
}
   catch (PDOException $e)
      { echo('Erreur de connexion: ' . $e->getMessage());
       exit();}
  $mybd=null;
?>
```

Exemple de requête SELECT

```
  <?php
try
      {
        $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');
        echo ('affichage du contenu de la table etudiants') . '<br />';
  $stmt1 = $mybd->prepare("select * from Etudiants where idProgramme
like ? ");
//liaison et affectation des valeurs au paramètre
$stmt1->bindParam(1, $idProgrog);
$idProgrog = 420;
//execution
 $stmt1->execute();
         while ($donnees = $stmt1->fetch())
            {
         echo $donnees[0] . $donnees[1] .'<br />';
            }
echo ($stmt1->rowCount()); // nombre totale de lignes
$stmt1->closeCursor();// fermer le curseur
      }
   catch (PDOException $e)
      {
       echo('Erreur de connexion: ' . $e->getMessage());
         exit();
      }
  $mybd=null;
?>
```

Remarque :

PDOStatement::bindParam: Associe une valeur à un paramètre

public bool **PDOStatement::bindParam** ($parameter , $value ,data_ype)

- Parameter : Identifiant du paramètre. Pour une requête préparée ou une procédure stockée. Le paramètre est l'indice du ? ou :NomParame. Lorsqu'indexé numériquement, l'index du premier paramètre commence à 1. (Pas à zéro comme C#)
- Value : La valeur à associer au paramètre.
- data_type : le type du paramètre

Exemple :

$stm->bindParam(1, $nom, PDO::PARAM_STR,20);

Indique que le paramère 1 reçoit le contenue de la variable nom. Ce paramètre est de type string de longueur 20

Cas3 : *Appels de procédures stockées.*

Attention : 

Recommandée : Bonne pratique surtout pour vos requêtes SELECT. Revoir tous les avantages à coder en procédures stockées

Lorsqu'on appelle une procédure stockée on procède de la même manière qu'une requête paramétrée. On utilise le PDO Statement avec le la méthode prepare() sauf qu'au lieu qu'elle prenne comme paramètre la requête, on utilise un CALL de la procédure.

Il faudra, en plus de faire le bind des paramètres en IN, **préciser le type des paramètres en OUT.**

Exemple 1 : procédure en insertion

Le code suivant va appeler la procédure **insertEtudiant** de la page 21 on rappelle que cette procédure prend 4 paramètres. Tous les paramètres sont en IN

Vous remarquerez que lorsque la procédure stockée a ses paramètres uniquement en IN alors la seule différence qu'il y a avec une requête paramétrée est la suivante :

Pour une requête paramétérée, dans la methode prepare(), c'est la requête qui est passée en paramètre.

Pour une procédure stockée, dans la methode prepar() ,c'est l'appel de la procédure (call) qui est en paramètre.

```php
<?php
try
{
      $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');
   echo ('insertion avec procedures') . '<br />';
// appel de la procédure
 $stmt1 = $mybd->prepare("CALL insertEtudiant(?,?,?,?)");
//Liaison des paramètres de la procédure.
$stmt1->bindParam(1, $id);
$stmt1->bindParam(2, $nom);
$stmt1->bindParam(3, $prenom);
$stmt1->bindParam(4, $codeDep);
// affectation de valeurs aux paramètres
$id =200;
$nom = 'Yacoub';
$prenom = 'Saliha';
$codeDep = 420;
// executer la requête
$total= $stmt1->execute();
echo('total insertion est ' . $total);
   }
    catch (PDOException $e)
            { echo('Erreur de connexion: ' . $e->getMessage());
          exit();}
  $bdd=null;
?>
```

Exemple 2, appel d'une procédure qui retourne un ensemble d'enregistrements. La procédure a un paramètre en IN. Celle de la page 22

```
try
{
      $mybd  = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');


      $stmt1 = $mybd ->prepare("call afficherEtudiantProgramme(?)",
array(PDO::ATTR_CURSOR, PDO::CURSOR_FWDONLY));
//Lier le paramètres en IN
$stmt1->bindParam(1, $nomprog , PDO::PARAM_STR, 30);
// affectation de valeurs aux paramètres
$nomprog = 'informatique';
// executer la requête
 $stmt1->execute();
//lecture des données et affichage
      while ($donnees = $stmt1->fetch())
      {
         echo $donnees[0] . $donnees[1] .'<br />';
      }
echo ($stmt1->rowCount());
$stmt1->closeCursor();// nombre totale de lignes
   }
    catch (PDOException $e)
      {   echo('Erreur de connexion: ' . $e->getMessage());
              exit();}
  $mybd =null;
?>
```

Exemple 3, appel d'une fonction qui retourne une valeur . Ce cas revient à exécuter une requête SELECT avec paramètre . La fonction appelée a un paramètre en IN et retourne un INT. C'est la fonction de la page 21

```
<?php
try
{
      $mybd = new
PDO('mysql:host=167.114.152.54;dbname=dbuser1;charset=utf8', 'user1',
'user1');


  $stm = $mybd->prepare("select compterEtudiantProgramme(?)" );
  $stm->bindParam(1, $pcode);
  $pcode=420;
  $stm->execute();
 if ($donnees = $stm->fetch())
      {
      echo $donnees[0] . '<br />';
      }
$stm->closeCursor();
   }
    catch (PDOException $e)
      {     echo('Erreur de connexion: ' . $e->getMessage());
              exit();}
  $bdd=null;
?>
```

Compléments PDO

PDOStatement::fetch — Récupère la ligne suivante d'un jeu de résultats PDO

Description ¶

public mixed **PDOStatement::fetch** ([int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0]]])

Récupère une ligne depuis un jeu de résultats associé à l'objet *PDOStatement*. Le paramètre fetch_style détermine la façon dont PDO retourne la ligne.

Liste de paramètres ¶

fetch_style : Contrôle comment la prochaine ligne sera retournée à l'appelant. Cette valeur doit être une des constantes *PDO::FETCH_ **, et par défaut, vaut la valeur de *PDO::ATTR_DEFAULT_FETCH_MODE* (qui vaut par défaut la valeur de la constante *PDO::FETCH_BOTH*).

PDO::FETCH_ASSOC: retourne un tableau indexé par le nom de la colonne comme retourné dans le jeu de résultats

PDO::FETCH_BOTH (défaut): retourne un tableau indexé par les noms de colonnes et aussi par les numéros de colonnes, commençant à l'index 0, comme retournés dans le jeu de résultats

PDO::FETCH_BOUND: retourne **TRUE** et assigne les valeurs des colonnes de votre jeu de résultats dans les variables PHP à laquelle elles sont liées avec la méthode PDOStatement::bindColumn()

PDO::FETCH_CLASS: retourne une nouvelle instance de la classe demandée, liant les colonnes du jeu de résultats aux noms des propriétés de la classe. Si fetch_style inclut PDO::FETCH_CLASS (c'est-à-dire *PDO::FETCH_CLASS* | *PDO::FETCH_CLASSTYPE*), alors le nom de la classe est déterminé à partir d'une valeur de la première colonne.

PDO::FETCH_INTO : met à jour une instance existante de la classe demandée, liant les colonnes du jeu de résultats aux noms des propriétés de la classe

PDO::FETCH_LAZY : combine *PDO::FETCH_BOTH* et *PDO::FETCH_OBJ*, créant ainsi les noms des variables de l'objet, comme elles sont accédées

PDO::FETCH_NAMED : retourne un tableau de la même forme que *PDO::FETCH_ASSOC*, excepté que s'il y a plusieurs colonnes avec les mêmes noms, la valeur pointée par cette clé sera un tableau de toutes les valeurs de la ligne qui a ce nom comme colonne

PDO::FETCH_NUM : retourne un tableau indexé par le numéro de la colonne comme elle est retourné dans votre jeu de résultat, commençant à 0

PDO::FETCH_OBJ : retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats

cursor_orientation :Pour un objet PDOStatement représentant un curseur scrollable, cette valeur détermine quelle ligne sera retournée à l'appelant. Cette valeur doit être une des constantes *PDO::FETCH_ORI_**, et par défaut, vaut *PDO::FETCH_ORI_NEXT*. Pour demander un curseur scrollable pour votre objet PDOStatement, vous devez définir l'attribut *PDO::ATTR_CURSOR* à *PDO::CURSOR_SCROLL* lorsque vous préparez la requête SQL avec la fonction PDO::prepare().

Offset : Pour un objet PDOStatement représentant un curseur scrollable pour lequel le paramètre *cursor_orientation* est défini à *PDO::FETCH_ORI_ABS*, cette valeur spécifie le numéro absolu de la ligne dans le jeu de résultats qui doit être récupérée.

Pour un objet PDOStatement représentant un curseur scrollable pour lequel le paramètre *cursor_orientation* est défini à *PDO::FETCH_ORI_REL*, cette valeur spécifie la ligne à récupérer relativement à la position du curseur avant l'appel à la fonction **PDOStatement::fetch()**.

Valeurs de retour : La valeur retournée par cette fonction en cas de succès dépend du type récupéré. Dans tous les cas, **FALSE** est retourné si une erreur survient.

La classe PDOStatement

PDOStatement->bindColumn — Lie une colonne à une variable PHP

PDOStatement->bindParam — Lie un paramètre à un nom de variable spécifique

PDOStatement->bindValue — Associe une valeur à un paramètre

PDOStatement->closeCursor — Ferme le curseur, permettant à la requête d'être de nouveau exécutée

PDOStatement->columnCount — Retourne le nombre de colonnes dans le jeu de résultats

PDOStatement->debugDumpParams — Détaille une commande préparée SQL

PDOStatement->errorCode — Récupère le SQLSTATE associé lors de la dernière opération sur la requête

PDOStatement->errorInfo — Récupère les informations sur l'erreur associée lors de la dernière opération sur la requête

PDOStatement->execute — Exécute une requête préparée

PDOStatement->fetch — Récupère la ligne suivante d'un jeu de résultat PDO

PDOStatement->fetchAll — Retourne un tableau contenant toutes les lignes du jeu d'enregistrements

PDOStatement->fetchColumn — Retourne une colonne depuis la ligne suivante d'un jeu de résultats

PDOStatement->fetchObject — Récupère la prochaine ligne et la retourne en tant qu'objet

PDOStatement->getAttribute — Récupère un attribut de requête

PDOStatement->getColumnMeta — Retourne les métadonnées pour une colonne d'un jeu de résultats

PDOStatement->nextRowset — Avance à la prochaine ligne de résultats d'un gestionnaire de lignes de résultats multiples

PDOStatement->rowCount — Retourne le nombre de lignes affectées par le dernier appel à la fonction PDOStatement::execute()

PDOStatement->setAttribute — Définie un attribut de requête

PDOStatement->setFetchMode — Définit le mode de récupération par défaut pour cette requête

La classe PDO

PDO::beginTransaction — Démarre une transaction

PDO::commit — Valide une transaction

PDO::__construct — Crée une instance PDO qui représente une connexion à la base

PDO::errorCode — Retourne le SQLSTATE associé avec la dernière opération sur la base de données

PDO::errorInfo — Retourne les informations associées à l'erreur lors de la dernière opération sur la base de données

PDO::exec — Exécute une requête SQL et retourne le nombre de lignes affectées

PDO::getAttribute — Récupère un attribut d'une connexion à une base de données

PDO::getAvailableDrivers — Retourne la liste des pilotes PDO disponibles

PDO::inTransaction — Vérifie si nous sommes dans une transaction

PDO::lastInsertId — Retourne l'identifiant de la dernière ligne insérée ou la valeur d'une séquence

PDO::prepare — Prépare une requête à l'exécution et retourne un objet

PDO::query — Exécute une requête SQL, retourne un jeu de résultats en tant qu'objet PDOStatement

PDO::quote — Protège une chaîne pour l'utiliser dans une requête SQL PDO

PDO::rollBack — Annule une transaction

PDO::setAttribute — Configure un attribut PDO

Sources

http://php.net/manual/fr/book.pdo.php

http://nl3.php.net/manual/fr/pdostatement.fetch.php

http://downloads.mysql.com/docs/refman-5.0-fr.pdf

http://php.net/manual/fr/pdostatement.bindvalue.php

https://dev.mysql.com/doc/refman/5.7/en/stored-programs-logging.html

http://php.net/manual/fr/security.database.sql-injection.php

https://blogs.oracle.com/svetasmirnova/entry/how_to_raise_error_in