Chapitre IV : MySQL

1. Introduction

MySQL est un gestionnaire de base de données (SGBD) libre. Il est très utilisé dans les projets libres et dans le milieu industriel. Il est supporté pratiquement par l'ensemble des systèmes d'exploitation : Linux, Windows, OS/2, freeBSD, ...

MySQL fait partie du quatuor LAMP : Linux, Apache, MySQL, PHP qui forment un outil puissant pour mettre en oeuvre des sites Web. Le couple PHP/MySQL est très utilisé par les sites Web et proposé par la majorité des hébergeurs.

Dans la suite de ce document, la sensibilité à la casse des commandes MySQL dépend du système d'exploitation. Ainsi, il est préférable de faire attention aux caractères majuscules et minuscules lors de l'écriture des commandes MySQL.

2. La Base de données

2.1 Création

Pour créer une nouvelle base de données, nous avons la syntaxe :

```
CREATE DATABASE `nom_bd`;
CREATE DATABASE IF NOT EXISTS `nom_bd`;
```

La première génère une erreur si la base de données existe déjà ce qui est évité par la seconde.

2.2 Suppression

Pour supprimer une base de données, nous avons la syntaxe :

```
DROP DATABASE `nom_bd`;
DROP DATABASE IF EXISTS `nom_bd`;
```

La première génère une erreur si la base de données n'existe pas ce qui est évité par la seconde.

2.3 Visualisation

Pour visualiser les bases de données existantes, nous avons la syntaxe :

```
SHOW databases;
```

2.4 Base de données par défaut

Pour définir la base de données par défaut, nous avons la syntaxe :

```
USE `nom_bd`;
```

Toutes les opérations futures porteront automatiquement sur cette base de données.

3. Les Tables (Relations)

3.1 Création

Pour créer une nouvelle table, nous avons la syntaxe :

```
CREATE TABLE `nom_table` (liste_attributes);
```

Les attributs sont définis sous forme de liste séparés par des virgules. Chaque attribut est donné comme suit :

```
`nom_attribut` type_attribut options
```

Le type est obligatoire par contre les options sont facultatives. Ainsi un attribut peut avoir 0 ou plusieurs options.

Exemple

On veut créer la table :

etudiant (mat, nom, prenom, DateNais, LieuNais, adresse, AnneeEtude)

```
CREATE TABLE `etudiant` (`mat` int(8) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, `nom`
varchar(30), `prenom` varchar(30), `DateNais` date, `LieuNais` varchar(20), `adresse`
varchar(50), `AnneeEtude` int(1) DEFAULT '1' NOT NULL );
```

3.2 Types d'attributs

Types numériques

Type	Borne inférieure	borne supérieure
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT*	-2147483648	2147483647
INT* UNSIGNED	0	4294967295
BIGINT	-9223372036854775808	9223372036854775807
BIGINT UNSIGNED	0	18446744073709551615

Les autres types

Type	Description
CHAR(M)	Chaîne de taille fixée à *M*, où 1<*M*<255, complétée avec des espaces si nécessaire.
CHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
VARCHAR(M)	Chaîne de taille variable, de taille maximum *M*, où 1<*M*<255.
VARCHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
TINYTEXT	Longueur maximale de 255 caractères.
TEXT	Longueur maximale de 65535 caractères.
MEDIUMTEXT	Longueur maximale de 16777215 caractères.
LONGTEXT	Longueur maximale de 4294967295 caractères.
DECIMAL(M,D)	Simule un nombre flottant de *D* chiffres après la virgule et de *M* caractères au maximum. Chaque chiffre ainsi que la virgule et le signe - (pas le +) occupe un caractère.
DATE	Date au format anglophone : *AAAA-MM-JJ*.
DATETIME	Date et heure au format anglophone : *AAAA-MM-JJ HH:MM:SS*.
TIMESTAMP	Affiche la date et l'heure sans séparateur : *AAAAMMJJHHMMSS*.
TIMESTAMP(M)	Idem mais *M* vaut un entier pair entre 2 et 14. Affiche les *M* premiers caractères de TIMESTAMP.
TIME	Heure au format *HH:MM:SS*.
YEAR	Année au format *AAAA*.

Type énumération

Un attribut de type **ENUM** peut prendre une valeur parmi celles définies lors de la création de la table plus la chaîne de caractères vide ainsi que **NULL** si la définition le permet. Les valeurs d'une énumération sont exclusivement des chaînes de caractères. Une énumération peut contenir 65535 éléments au maximum.

```
`nom_attribut` ENUM("valeur1","valeur2",… , "valeur_n")
`nom_attribut` ENUM("valeur1","valeur2",… , "valeur_n") NULL
```

Type ensemble

Un attribut de type **SET** peut prendre pour valeur la chaîne de caractères vide, **NULL** ou une chaîne contenant une liste de valeurs qui doivent être déclarées lors de la définition de l'attribut. Il ne peut être défini que 64 éléments au maximum.

Par exemple : **`lieu` SET("foret", "montagne", "plage") NOT NULL**

L'attribut **lieu** peut prendre les valeurs suivantes : **""** (chaîne vide), **"forêt, montagne"**, **"plage, forêt, montagne"** ou toute autre combinaison des trois valeurs.

`lieu` SET("foret", "montagne", "plage") NULL : Peut prendre, en plus de celles précédentes, la valeur **NULL**.

3.2.1 Options des attributs

Le tableau ci-dessous résume les principales options qu'on peut affecter à un attribut dans MySQL.

Option	Description
NOT NULL ou **NULL**	Spécifie si une valeur **NULL** peut être stockée dans le champ (champ vide ou non).
DEFAULT "val"	Spécifie une valeur à entrer lorsqu'aucune valeur n'est spécifiée.
AUTO_INCREMENT	Concerne uniquement les valeurs numériques entières, la valeur est incrémentée de 1 à chaque insertion d'un nouvel enregistrement. Il ne peut y avoir qu'une seule valeur **AUTO_INCREMENT** par table. On l'utilise généralement comme clé primaire.
PRIMARY KEY	Définit l'attribut comme clé primaire de cette table.
FOREIGN KEY	Définit une clé étrangère.
KEY ou **INDEX**	Définit un index multiple. Les index permettent d'accélérer les recherches.
UNIQUE	Définit un index unique, il ne peut pas y avoir deux fois la même valeur dans le champ.

3.2.2 Clé primaire :

Lors de la création d'une table, souvent il est toujours nécessaire d'avoir des éléments permettant d'identifier de manière unique chaque occurrence (enregistrement) : c'est la clé primaire. Celle-ci doit avoir les propriétés **NOT NULL**, **UNIQUE** et peut être même **AUTO_INCREMENT**. Pour simplifier, l'utilisation de l'option **PRIMARY KEY** remplacera **NOT NULL** et **UNIQUE**.

La clé primaire peut être associée simultanément à plusieurs attributs. Dans ce cas, la syntaxe sera différente. Il faut rajouter juste après la définition du dernier attribut et avant la parenthèse fermante de la création de la table : **PRIMARY KEY (`nom_attribut_1`, `nom_attribut_2`, … , `nom_attribut_n`)**.

3.3 Suppression

La commande **DROP TABLE** prend en paramètre le nom de la table à supprimer. Toutes les données qu'elle contient seront supprimées ainsi que sa définition.

```
DROP TABLE `nom_table`;
DROP TABLE IF EXISTS `nom_table`;
```

La première génère une erreur si la table n'existe pas ce qui est évité par la seconde.

3.4 Visualisation

Pour lister les tables (relations) qui existent dans une base de données, on utilise la commande suivante :

```
SHOW TABLES;
```

Pour visualiser les colonnes (attributs) d'une table, nous utilisons la commande suivante :

```
SHOW COLUMNS FROM `nom_table`;
```

3.5 Modification

Il est possible de modifier la définition d'une table à tout moment par la commande **ALTER TABLE** pour : ajouter/supprimer un attribut, créer/supprimer une clé primaire, ajouter une contrainte d'unicité (interdire les doublons), changer la valeur par défaut d'un attribut, changer totalement la définition d'un attribut, changer le nom de la relation, ...

3.5.1 Ajout d'un attribut

Pour ajouter un nouvel attribut, on doit préciser son nom et éventuellement sa position parmi les attributs qui existent dans la table. La syntaxe est la suivante :

```
ALTER TABLE `nom_table` ADD `nom_nouveau_attribut` type options;
```

Rajoute le nouvel attribut à la dernière position.

```
ALTER TABLE `nom_table` ADD `nom_nouveau_attribut` type options FIRST;
```

Rajoute le nouvel attribut à la première position.

```
ALTER TABLE `nom_table` ADD `nom_nouveau_attribut` type options AFTER `nom_attribut`;
```

Rajoute le nouvel attribut en l'insérant juste après l'attribut **nom_attribut**.

Exemple :

Ajouter l'attribut nationalité qui est une chaîne représentant un nombre de 3 chiffres:

```
ALTER TABLE `etudiant` ADD `nationalite` DECIMAL(3,0) AFTER `prenom`;
```

3.5.2 Suppression d'un attribut

Pour supprimer un attribut, on doit juste préciser son nom. La syntaxe est la suivante :

```
ALTER TABLE `nom_table` DROP `nom_attribut`;
```

Exemple :

Supprimer l'attribut (colonne) telephone de la table etudiant :

```
ALTER TABLE `etudiant` DROP `telephone`;
```

3.5.3 Suppression d'une clé primaire :

Pour modifier la clé primaire d'une table, il faut d'abord supprimer l'ancienne clé. Cette suppression n'engendre pas la suppression de l'attribut. La syntaxe est la suivante :

```
ALTER TABLE `nom_table` DROP PRIMARY KEY;
```

Remarque : Ici, nous n'avons pas besoin de préciser le nom de l'attribut qui est clé primaire.

3.5.4 Création d'une clé primaire

La création d'une clé primaire n'est possible qu'en l'absence de clé primaire dans la relation. La syntaxe est :

```
ALTER TABLE `nom_table` ADD PRIMARY KEY (`attribut`);
ALTER TABLE `nom_table` ADD PRIMARY KEY (`attribut_1`, `attribut_2`, …, `attribut_n`);
```

3.5.5 Modifier un attribut

Pour changer la définition de l'attribut sans le renommer :

```
ALTER TABLE `nom_table` MODIFY `nom_attribut` type options;
```

Pour changer sa définition en le renommant :

```
ALTER TABLE `nom_table` CHANGE `ancien_nom_attribut` `nouveau_nom_attribut` type options;
```

Les options ne sont pas obligatoires lors de la modification d'un attribut.

4. Les enregistrements

4.1 Ajout (Insertion)

Pour ajouter un enregistrement dans une table, il existe plusieurs façons de faire :

4.1.1 Insertion étendue

La syntaxe est la suivante :

```
INSERT INTO `nom_table` (`att_1`, `att_2`, …, `att_n`) VALUES ("val_1", "val_2", …, "val_n");
```

Il n'est pas obligatoire de donner la liste des attributs dans l'ordre ou de donner des valeurs à tous les attributs.

4.1.2 Insertion standard

La syntaxe est la suivante :

```
INSERT INTO `nom_table` VALUES("val_1", "val_2", …, "val_n");
```

La liste des valeurs doit être exhaustive (nombre de valeurs égale au nombre d'attributs) et ordonnée selon la définition des attributs de la table.

4.1.3 Insertion complète :

Cette façon permet d'insérer plusieurs enregistrements en une seule requête. La syntaxe est la suivante :

```
INSERT INTO nom_table VALUES (liste 1 de val), (liste 2 de val), …, (liste n de val);
```

Remarque : L'insertion complète et l'insertion étendue peuvent être associées dans une même requête.

4.2 Modification

Pour modifier un ou plusieurs enregistrements dans une table :

```
UPDATE `relation` SET `attribut`="valeur", … [ WHERE condition ] [LIMIT a ]
```

Exemple :

```
UPDATE `etudiant` SET `AnetEtude` = `AnetEtude` + 1 WHERE `AnetEtude` < "3"
UPDATE `etudiant` SET `DateNais`="12-02-1983", `LieuNais`="Alger" WHERE `Mat`= "200600001234";
```

4.3 Suppression

Attention la suppression est définitive et sans confirmation.

```
DELETE  FROM `relation` [ WHERE condition ] [ LIMIT a ];
DELETE  FROM `etudiant` WHERE `nom`="Ait Slimane" AND `prénom`="Ali";
```

Pour vider entièrement une table, il suffit de ne pas mettre de clause **WHERE**.

```
DELETE FROM `Personnes`;
```

5. Optimisation d'une table

L'optimisation consiste à réorganiser physiquement les données d'une table. En effet, suite aux différentes suppressions dans une table, les index des enregistrements supprimés sont conservés, rallongeant d'autant les sélections. Pour supprimer ces index obsolètes, il faut l'optimiser.

```
OPTIMIZE TABLE `Relation`;
OPTIMIZE TABLE `etudiant`;
```

6. Sélection - Recherche

La syntaxe générale d'une requête de sélection qui permet d'effectuer une recherche sur une ou plusieurs tables est la suivante :

```
SELECT [ DISTINCT ] attributs [ INTO OUTFILE fichier ]
[ FROM relation ]
[ WHERE condition ]
[ GROUP BY attributs [ ASC | DESC ] ]
[ HAVING condition ]
[ ORDER BY attributs ]
[ LIMIT [a,] b ];
```

Les éléments de la requête de sélection sont :

Nom	Description
SELECT	Spécifie les attributs dont on souhaite connaître les valeurs. On met ***** si on sélectionne tous les attributs.
DISTINCT	Permet d'ignorer les doublons de lignes de résultat.
INTO OUTFILE	Spécifie le fichier sur lequel effectuer la sélection.
FROM	Spécifie le ou les relations sur lesquelles effectuer la sélection.
WHERE	Définit le ou les critères de sélection (sous forme de condition) sur des attributs.
GROUP BY	Permet de grouper les lignes du résultat selon un ou plusieurs attributs.
HAVING	Définit un ou plusieurs critères de sélection (sous forme de condition) sur des ensembles de valeurs d'attributs après groupement. Elle est presque identique à **WHERE** sauf qu'elle nécessite **GROUP BY** et on peut y insérer des fonctions MySQL.
ORDER BY	Permet de définir l'ordre (**ASC** ascendant par défaut ou **DESC** descendant) dans l'envoi des résultats.
LIMIT	Permet de limiter le nombre de lignes du résultat. **LIMIT 10** permet de limiter la recherche sur les 10 résultats qui suivent les 20 premiers, c'est-à-dire du 21ème au 30ème. **LIMIT 20, 10** permet de limiter la recherche sur les 10 premiers résultats. Elle nécessite la clause **ORDER BY**.

6.1 Conditions

Les conditions MySQL sont écrites dans le **WHERE** ou le **HAVING**. Il est possible d'utiliser :

- Les différents opérateurs de comparaisons : égalité **=**, différence **!=** ou **<>**, supérieur **>**, inférieur **<**, supérieur ou égal **>=**, inférieur ou égal **<=**.
- L'opérateur de comparaison **LIKE**. Il permet de comparer une chaîne de caractère par rapport à un modèle où on peut utiliser **%** pour indiquer qu'il s'agit d'une sous-chaîne de caractère quelconque ou **_** qui représente un seul caractère quelconque.
- L'opérateur **BETWEEN** pour définir l'appartenance à un intervalle qui peut être nombre, chaîne ou date.
- Les opérateurs logiques **AND** et **OR**.
- L'opérateur ensembliste **IN** pour définir l'appartenance à un ensemble de valeurs. Son principal avantage est de permettre d'éviter d'écrire une condition avec plusieurs **OR**.
- L'opérateur ensembliste **NOT IN** pour définir la non appartenance.
- Les opérateurs **IS NULL** ou **IS NOT NULL** pour vérifier si la valeur d'un attribut est vide ou non.

Exemples :

`email` LIKE '%usthb.dz' : les valeurs de l'attribut email qui se terminent par usthb.dz

`niv` LIKE 'L_' : Les valeurs qui commencent par *L* suivi d'un caractère sont acceptées. Par exemple :*L1*, *L2*, *L3*...

`date_naiss` BETWEEN '1999-01-01' AND '2001-12-31' : Pour les personnes qui sont nées entre 1999 et 2001

`niv` IN ('L1','L2','L3') : équivalent à écrire **`niv`='L1' OR `niv`='L2' OR `niv`='L3'**

6.2 Fonctions MySQL

On peut faire appel à certaines fonctions prédéfinies dans les requêtes MySQL. Les fonctions peuvent être écrites après la clause **SELECT** ou dans la condition qui suit la clause **HAVING**. Le tableau ci-dessous donne quelques MySQL.

Fonction	Description
COUNT()	Donne le nombre de lignes dans un résultat.
SUM()	Calcule la somme.
AVG()	Retourne la moyenne.
MAX()	Retourne le maximum.
MIN()	Retourne le minimum.
ROUND()	Permet d'arrondir la valeur.
UPPER()	Convertit une chaîne de caractères en majuscule.
LOWER()	Convertit une chaîne de caractères en minuscule.
CONCAT()	Réalise la concaténation de chaînes de caractères.
LENGTH()	Retourne le nombre de caractères dans une chaîne de caractères.
CURRENT_DATE()	Retourne la date actuelle.
NOW()	Retourne la date et heure actuelle.

6.3 Exemples :

Pour extraire de votre base de données des informations, comme la liste des étudiants nés à un endroit dont le nom contient le mot « oued ».

```
SELECT `nom`, `prenom`
FROM `etudiant`
WHERE `LieuNais` LIKE "%oued%";
```

Pour lister les noms et prénoms des étudiants de la troisième année :

```
SELECT `nom`, `prénom`
FROM `etudiant`
WHERE `anet`= '3';
```

Pour trouver les noms, prénoms et moyennes des étudiants par ordre de mérite (ordonnés dans un ordre descendant de la moyenne) :

```
SELECT `nom`, `prénom`, `moy`
FROM `etudiant`
ORDER BY `moy` DESC;
```

Pour donner pour chaque client le produit acheté :

```
SELECT `client.nom`, `produit.nom`
FROM `client`, `produit`
WHERE `client.id` = `produit.id_client`
ORDER BY `client.nom`
LIMIT 0, 10; #même chose que LIMIT 10
```

Cette dernière requête effectue une sélection sur deux tables « *client* » et « *produit* ». Le nom du client (*client.nom*) est récupéré à travers l'attribut *nom* de la table « *client* » et le nom du produit (*produit.nom*) est récupéré à travers l'attribut *nom* de la table « *produit* ». Les résultats sont triés par ordre alphabétique des noms des clients et leur nombre est limité à 10

Le symbole **#** permet d'insérer un commentaire dans MySQL. On peut aussi utiliser le point-virgule (**;**).

Pour calculer le nombre d'enregistrements (lignes) dans la table « *etudiant* » :

```
SELECT COUNT(*)
FROM `etudiant`;
```

Pour retourner pour chaque produit la moyenne des prix d'achats :

```
SELECT `nom_prod`, AVG(`prix_achat`)
FROM `Commande`
GROUP BY `nom_produit`;
```

Pour trouver le nom et la somme des achats pour chaque client ayant un total d'achat > 10 000 :

```
SELECT `nom_client`, SUM(`prix_achat`)
FROM `Commande`
GROUP BY `nom_client`
HAVING SUM(`prix_achat`)>10000;
```

Pour récupérer les noms et prénoms des étudiants dont le matricule est sur moins de 12 positions :

```
SELECT `nom`, `prenom`
FROM `etudiant`
WHERE LENGTH(`mat`)< 12;
```

6.4 Combinaison de requêtes

Il est possible d'utiliser des opérateurs ensemblistes pour combiner des requêtes de sélection. En fin de compte, une opération ensembliste est réalisée sur les ensembles de résultats des requêtes.

6.4.1 L'union

L'opérateur d'union dans MySQL est **UNION**. Cet opérateur permet de réaliser l'union entre les ensembles de résultats des deux requêtes. La syntaxe générale est : **`Requête1 UNION Requête2;`**

Cette combinaison donne comme résultat : $Ens\_Rés = Ens\_Rés(Requête1) \cup Ens\_Rés(Requête2)$.

L'exemple ci-dessous donne tous les étudiants des deux facultés :

```
SELECT ` nom`, `prenom`
FROM `FI`
UNION
SELECT `nom`, `prenom`
FROM `FGC`;
```

Remarque : Les attributs doivent être au même nombre, de même type et respectant le même ordre.

6.4.2 L'intersection

MySQL n'a pas un opérateur spécifique pour réaliser l'intersection comme c'est le cas pour d'autres langages. Tout de même, il est possible de réaliser une intersection grâce à l'opérateur ensembliste **IN**.

L'exemple ci-dessous nous donne les noms des entreprises qui sont en même temps importatrices et exportatrices :

```
SELECT `nom_entreprise`
FROM `importateur`
WHERE
`nom_entreprise` IN (  SELECT `nom_entreprise`
                       FROM `exportateur`);
```

Remarque : L'intersection réalisée à l'aide de **IN** peut générer des doublons dans les résultats. Pour résoudre ce problème, on rajoute la clause **DISTINCT**.

6.4.3 La soustraction

L'opérateur de soustraction dans MySQL est **MINUS**. Cet opérateur permet de donner les résultats de la première requête qui n'existent pas dans les résultats de la seconde requête. La syntaxe générale est : **Requête1 MINUS Requête2;**

Cette combinaison donne comme résultat : *Ens_Rés = Ens_Rés(Requête1) - Ens_Rés(Requête2)*.

L'exemple ci-dessous donne tous les clients qui n'ont pas effectué d'achats :

```
SELECT `nom_client`
FROM `client`
MINUS
SELECT `nom_client`
FROM `achat`;
```

Remarque : Les attributs doivent être au même nombre, de même type et respectant le même ordre.

6.5 Les jointures

La jointure permet de combiner les résultats deux requêtes faites sur deux tables différentes sur la base d'un attribut commun. Il existe plusieurs types de jointures dans MySQL.

6.5.1 Jointure interne

La jointure interne se fait à l'aide de l'opérateur **INNER JOIN**. Elle permet de formuler une requête combinée sur deux tables grâce à un attribut donné dans la clause **ON**. Les résultats retournés sont disponibles sur les deux tables en même temps.

Par exemple, si on veut sélectionner les étudiants de la 3ème année et leurs moyennes en 3ème année. Les noms des étudiants sont disponibles sur la table « *etud* » et les moyennes sont disponibles sur la table « *delib* ».

```
SELECT `etud.nom`, etud.prenom`, `delib.moy`
FROM `etud`
INNER JOIN `delib`
ON (`etud.mat`=`delib.mat_etud`)
WHERE `delib.anet`=3;
```

6.5.2 Jointure gauche / jointure droite

La jointure gauche (appelée aussi jointure externe gauche) se fait à l'aide de l'opérateur **LEFT JOIN**. Elle retourne les enregistrements de la première table même si la condition n'est pas vérifiée pour la seconde. C'est-à-dire les enregistrements de la première table qui n'ont pas de correspondance dans la seconde table sont retournés. Dans ce cas les valeurs des attributs de la seconde table seront à **NULL**.

La jointure droite (appelée aussi jointure externe droite) se fait à l'aide de l'opérateur **RIGHT JOIN**. Elle retourne les enregistrements de la seconde table même si la condition n'est pas vérifiée pour la première. C'est-à-dire les enregistrements de la seconde table qui n'ont pas de correspondance dans la première table sont retournés. Dans ce cas les valeurs des attributs de la première table seront à **NULL**.

La syntaxe de ces deux jointures est identique à la jointure interne (voir section précédente) sauf le mot **INNER** qui est remplacé soit par **LEFT**, soit par **RIGHT**.

6.5.3 Jointure naturelle

La jointure naturelle se fait par rapport à un attribut commun entre deux table et à l'aide de l'opérateur **NATURAL JOIN**. Les deux tables doivent avoir un attribut qui porte le même nom et de même type. Sans préciser l'attribut commun dans la requête, la jointure est faite naturellement sur cet attribut. La syntaxe est identique aux jointures précédentes sauf que la clause **ON** n'est pas utilisée dans ce cas.

6.5.4 Jointure de croisement

La jointure de croisement se fait à l'aide de l'opérateur **CROSS JOIN**. Elle retourne le produit cartésien de deux tables. Si on a *m* colonnes et *n* lignes dans la table 1 et on a *q* colonnes et *p* lignes dans la table 2, on obtient dans le résultat *m+q* colonnes et *n*p* lignes. C'est-à-dire toutes les combinaisons possibles entre chaque ligne de la première et chaque ligne de la seconde. La syntaxe est identique à la jointure naturelle.

6.6 Les alias

Un alias permet d'identifier de manière simplifiée une table ou un attribut grâce à la clause **AS**. Pour mieux comprendre le principe, nous donnons l'exemple suivant :

On a la table personne suivante : *Personne(id, nom, prenom, id_pere)*

Où l'identifiant du père est dans la table « *personne* ». A partir de cet identifiant, on peut avoir le nom du père est dans cette même table personne. On veut trouver pour chaque personne : son prénom, le prénom de son père. Pour ce, on doit faire une jointure entre la table « *personne* » et la table « *personne* ». La question qui se pose est : quelle est la première et quelle est la seconde? La solution consiste à utiliser les alias pour les distinguer.

```
Select p1.prenom f, p2.prenom p
From Personne AS p1
LEFT JOIN Personne AS p2
ON p1.id_pere = p2.id
ORDER BY p;
```

Pour la 1ère ligne on aurait pu écrire : **Select p1.prenom as f, p2.prenom as p**

Cette requête donne pour chaque personne son père. Ceux pour qui, on n'a pas l'information, il met **NULL** comme père (**LEFT JOIN**).

7. PHP et MySQL

Il existe quatre (4) API permettant de se connecter et communiquer avec MySQL. La première est la plus ancienne API (extension MySQL) qui est devenue obsolète depuis la version 5.5 du PHP et génère un message d'avertissement (warning) « *depricated function* ». Actuellement, il est recommandé d'utiliser soit l'extension MySQLi (le *i* pour le mot anglais *improved*, c'est-à-dire une version améliorée de l'API MySQL), soit l'extension PDO. Pour l'extension MySQLi, il existe le style procédural et le style orienté objet, alors que l'extension PDO est purement orientée objet.

Nous nous intéressons dans ce qui suit aux trois (3) dernières.

7.1 Connexion

Nous utilisons une connexion simple, lorsqu'il s'agit de créer une base de données. Lorsqu'il s'agit de manipuler des éléments (tables, enregistrements, recherche, …) de la base de données existante, nous utilisons une connexion avec sélection de la base de données.

7.1.1 MySQLi procédural

Pour se connecter à un serveur MySQL, nous utilisons la fonction **mysqli_connect**. Cette fonction est identique à celle du MySQL classique (voir section ci-dessus).

Syntaxe :

```
$connexion = mysqli_connect ($host, $login, $password);
```

La fonction nécessite trois paramètres :

- Le nom de l'hôte. Si MySQL est sur la même machine, on peut mettre **localhost**.
- Le nom d'utilisateur et le mot de passe du serveur.

Si on veut sélectionner la base de données on rajoutant un quatrième paramètre (**$bdd**) qui correspond au nom de la base de données.

```
$connexion = mysqli_connect ($host, $login, $password, $bdd);
```

7.1.2 MySQLi orienté objet

Pour se connecter à un serveur MySQL, nous devons créer une instance de l'objet **mysqli** à l'aide de la fonction **new**. L'objet créé ne doit pas être modifié ou supprimé jusqu'à la fermeture de la connexion.

Syntaxe :

```
$connexion = new mysqli($host, $login, $password);
```

Les trois paramètres ci-dessous sont nécessaires pour établir la connexion :

- Le nom de l'hôte. Si MySQL est sur la même machine, on peut mettre **localhost**.
- Le nom d'utilisateur et le mot de passe du serveur.

Si on veut sélectionner la base de données on rajoutant un quatrième paramètre (**$bdd**) qui correspond au nom de la base de données.

```
$connexion = new mysqli($host, $login, $password, $bdd);
```

7.1.3 PDO

Pour se connecter à un serveur MySQL, nous devons créer une instance de l'objet **PDO** à l'aide de la fonction **new**. L'objet créé ne doit pas être modifié ou supprimé jusqu'à la fermeture de la connexion.

Syntaxe :

```
$connexion = new PDO("mysql:host=$host;dbname=$bdd", $login, $password);
```

Les trois paramètres ci-dessous sont nécessaires pour établir la connexion :

- Le premier est une chaine de caractère composée de trois éléments : type de SGBD (dans notre cas, il s'agit de mysql), nom de l'hôte (Si MySQL est sur la même machine, on peut mettre **localhost**) et le nom de la base de données.
- Le nom d'utilisateur et le mot de passe du serveur.

7.2 Envoi d'une requête vers le serveur

Pour envoyer une requête, il faut l'écrire sous forme de chaîne de caractère. Il préférable de la sauvegarder dans une variable :

```
$req = "la requête…";
```

7.2.1 MySQLi procédural

On utilise la fonction **mysqli_query($connexion, $req)** pour transmettre la requête au serveur MySQL.

Cette fonction retourne les résultats sous forme d'un tableau à deux dimensions. Chaque ligne de ce tableau contient un résultat et chaque colonne un attribut. Ceci dans le cas où la requête consiste à une sélection (**SELECT**). **$resultat** sera faux si la requête ne trouve aucun résultat.

S'il s'agit d'un autre type de requête (création, modification ou suppression d'une base de données ou une table / insertion, modification ou suppression d'un enregistrement, …), on utilise la même fonction mais qui retourne un

booléen (vrai si l'opération a réussi). Dans ce cas, il est aussi possible d'appeler **mysqli_query** comme une procédure.

7.2.2 MySQLi orienté objet

On appelle la méthode **query($req);** qui crée une instance de l'objet **mysqli_results**. La syntaxe est la suivante :

```
$resultat=$connexion->query($req);
```

Grâce aux méthodes de la classe **mysqli_results**, on peut traiter et interpréter les résultats de la requête. En cas d'échec, elle retourne **FALSE**. Ceci est intéressent pour les requêtes de création, suppression, rajout, modification (autre que la sélection), afin de tester si la requête a réussi.

7.2.3 PDO

Dans l'API PDO, il faut distinguer les requêtes retournant des résultats (comme la sélection) des requêtes qui ne retournent pas de résultats (création, suppression, ajout, modification).

Pour le premier cas, on peut utiliser la méthode **query** ou la combinaison des méthodes **prepare** et **execute**. **query** est une méthode de la classe **PDO** qui envoie la requête et permet de créer une instance de l'objet **PDOStatement** qui sera utilisé par la suite pour traiter les résultats de la requête. **prepare** est une méthode de la classe **PDO** qui crée une instance de l'objet **PDOStatement**. Cette méthode n'envoie pas la requête mais juste la prépare pour la méthode et **execute** de la classe **PDOStatement**.

Ainsi on écrit soit :

```
$s=$connexion->query($req);
```

Soit :

```
$s=$connexion->prepare($req);
$s->execute();
```

Pour le second cas, on utilise la méthode **exec** de la classe **PDO** qui envoie la requête au SGBD. Ainsi, on écrit :

```
$connexion->exec($req);
```

7.3 Traitement des résultats d'une requête

Après l'envoi de la requête, le serveur répond et sa réponse sera interprétée selon l'API choisie.

7.3.1 MySQLi procédural

Les fonctions qui existent permettent de lire une ligne du tableau de résultats et font avancer le pointeur vers la ligne suivante. S'il n'y a plus de lignes à lire, elles retournent faux. La différence entre les fonctions est la forme du tableau retourné (indexé, associatif ou les deux en même temps).

La fonction **mysqli_fetch_row** retourne un tableau indexé (les clés sont des indices : 0, 1, 2, …). La fonction **mysqli_fetch_assoc** retourne un tableau associatif (les clés sont des attributs). La fonction **mysqli_fetch_array** retourne par défaut un tableau indexé et associatif en même temps. Les syntaxes sont les suivantes **:**

```
$ligne=mysqli_fetch_row($resultat);
$ligne=mysqli_fetch_assoc($resultat);
$ligne=mysqli_fetch_array($resultat, $type);
```

Le tableau **$ligne** est indexé dans le premier cas et associative pour le deuxième cas. Dans le troisième cas, il dépend du paramètre facultatif **$type** qui peut prendre l'une des valeurs prédéfinies suivantes : **MYSQLI_NUM** (la fonction est équivalente à **mysqli_fetch_row**), **MYSQLI_ASSOC** (la fonction est équivalente à **mysqli_fetch_assoc**) ou **MYSQLI_BOTH** qui est la valeur par défaut et qui retourne un tableau à la fois indexé et associatif.

Il est possible de consulter le nombre de résultats retournés par la requête à l'aide de la fonction **mysqli_num_rows($resultat)** qui retourne un entier.

7.3.2 MySQLi orienté objet

Il est possible de faire appel à l'une des trois méthodes de la classe **mysqli_results** : **fetch_row**, **fetch_assoc** ou **fetch_array**. Pour cela, nous exploitons l'objet **$resultat** créé par la méthode **query**. La syntaxe est la suivante :

```
$ligne=$resultat->fetch_row();
$ligne=$resultat->fetch_assoc();
$ligne=$resultat->fetch_array($type);
```

Ces trois méthodes se comportent exactement comme les trois fonctions de MySQLi procédural citées dans la section précédente.

Il est possible de consulter le nombre de résultats retournés par la requête à l'aide de l'attribut **num_rows** de la classe **mysqli_results** : qui retourne un entier. On écrit **$nb=$resultat->num_rows**.

7.3.3 PDO

Pour l'API PDO, il faut d'abord appeler la méthode **setFetchMode** de la classe **PDOStatement** afin de définir le type des résultats retournés (tableau associatif, objet, …). Par la suite, il faut utiliser la méthode **fetch** de la de la même classe pour lire une ligne et avancer. La syntaxe est la suivante :

```
$s->setFetchMode($type);
$ligne=$s->fetch();
```

Le **$type** peut prendre une des valeurs suivantes :

- **PDO::FETCH_ASSOC** retourne tableau associatif.
- **PDO::FETCH_NUM** retourne tableau indexé.
- **PDO::FETCH_BOTH** retourne les deux tableaux (associatif et indexé) en même temps.
- **PDO::FETCH_OBJ** retourne un objet où **$res->nom_attribut** représente une valeur.

7.4 Gestion des erreurs

Pour éviter tout problème et de continuer le script avec de faux résultats. Il est préférable dans le cas où la communication entre le PHP et le serveur MySQL n'aboutit pas d'arrêter le script et d'afficher un message d'erreur. Ceci est faisable à l'aide de la fonction **die** qui affiche le message d'erreur (transmis en paramètre) puis termine le script. La syntaxe suivante :

```
die("message d'erreur") ;
```

Il est aussi possible de récupérer le message (texte) d'erreur généré par le système ainsi que le code (numéro) de l'erreur et ce selon l'API choisie.

7.4.1 MySQLi procédural

Dans MySQLi procédural, il y a distinction entre l'erreur générée par une connexion et les erreurs générées par les autres manipulations.

La fonction qui retourne le message d'erreur suite à une connexion :

```
$msg = mysqli_connect_error();
```

La fonction qui retourne le code de l'erreur suite à une connexion :

```
$cd_err = mysqli_connect_errno();
```

La fonction qui retourne le message d'erreur :

```
$msg = mysql_error($connexion);
```

La fonction qui retourne le code de l'erreur :

```
$cd_err = mysql_errno($connexion);
```

7.4.2 MySQLi orienté objet

Dans MySQLi, les quatre fonctions citées dans la section ci-dessus sont représentées par des attributs (propriétés).

Le message d'erreur suite à une connexion :

```
$msg = c->connect_error;
```

Le code de l'erreur suite à une connexion :

```
$cd_err = c->connect_errno;
```

Le message d'erreur :

```
$msg = c->error;
```

Le code de l'erreur :

```
$cd_err = c->errno;
```

7.4.3 PDO

Pour l'API PDO, il y a distinction entre l'erreur générée par les méthodes **prepare** et **query** (qui créent un objet **PDOStatement**) et les erreurs générées par les autres manipulations.

La méthode qui retourne le message d'erreur (erreur générée par les méthodes **prepare** et **query**) :

```
$msg = s->errorInfo();
```

La méthode qui retourne le code de l'erreur (erreur générée par les méthodes **prepare** et **query**) :

```
$cd_err = s->errorCode;
```

La méthode qui retourne le message d'erreur (erreur générée par d'autres manipulations) :

```
$msg = c->errorInfo();
```

La méthode qui retourne le code de l'erreur (erreur générée par d'autres manipulations) :

```
$cd_err = c->errorCode;
```

7.5 Libérer l'espace mémoire

Il est nécessaire d'optimiser l'espace mémoire du serveur. Pour cela, il est utile de libérer l'espace où sont récupérés les résultats des requêtes.

7.5.1 MySQLi procédural

La libération de l'espace mémoire alloué aux résultats des requêtes se fait à l'aide de la fonction suivante :

```
mysqli_free_result($resultat);
```

7.5.2 MySQLi orienté objet

La libération de l'espace mémoire alloué aux résultats des requêtes se fait à l'aide de la méthode suivante :

```
$resultat->close();
```

7.6 Fermeture de la connexion

A la fin, il faut fermer la connexion avec le serveur.

7.6.1 MySQLi procédural

On utilise la variable **ressource** qu'on a utilisé au début pour l'ouverture de la connexion comme suit :

```
mysqli_close($connexion);
```

7.6.2 MySQLi orienté objet

On fait appel à la méthode **close** via l'objet créé lors de la connexion comme suit :

```
$connexion->close();
```

7.6.3 PDO

Il suffit d'affecter la valeur NULL à l'objet qui a été créé lors de la connexion comme suit :

```
$connexion = NULL;
```

7.7 Exemples

Dans ce qui suit, nous présentons quelques exemples qui nous permettent de manipuler les trois API vues ci-dessus.

7.7.1 Lister des étudiants à partir d'une table avec l'API MySQLi procédural

Ci-dessous, un exemple en MySQLi procédural où nous utilisons la majorité des fonctions définies ci-dessus :

```
$host='localhost';
$login='admin5';
```

```
$passwd='Top_Secret!';
$bdd='ma_base';
$c=mysqli_connect($host, $login, $psswd, $bdd);
if (!c) die("échec : " . mysqli_connect_error());
$q='SELECT `nom`, `prenom` FROM `Etudiant` WHERE `anet`="3"';
$r=mysqli_query($c,$q);
if (mysqli_num_rows($r)>0) {
 while($l=mysqli_fetch_assoc($r)) {
   echo 'Nom : ' . $l['nom '] . ' – ';
   echo 'Prénom : ' . $l['prénom '] . '.<br>';}
  mysqli_free_result($r);}
else {
   echo 'Aucun résultat'; }
mysqli_close($c);
```

7.7.2 Insertion de trois personnes avec les trois API

Dans ce qui suit, nous allons écrire pour chaque API deux scripts. Le premier, nommé *e.php*, consiste à donner la possibilité à un utilisateur d'insérer trois personnes dans la table *personne* (*id*, *nom*, *prenom*, *id_pere*) où *id_pere* est le *id* du père de cette personne qui existe déjà dans cette table. Le formulaire contient pour chaque personne deux champs textuels pour saisir le nom et le prénom de la personne ainsi qu'une boîte de liste contenant les prénoms de toutes les personnes qui existent dans la table afin de sélectionner le père. Ceci nécessite une requête select sur la table *personne*. Le second script, nommé *r.php* permet d'insérer les trois personnes tout en gérant les erreurs.

API MySQLi procédural

Script *e.php*

```
<html>
<head>
   <meta charset="utf-8" />
   <title>Exemple de BdD : MySQLi Procédural</title>
</head>
<body>
   <?php
      //Préparer l'affichaage dans des variables
      $n='Nom : <input type="text" name="nom[]" />. ';
      $f='Prénom : <input type="text" name="prenom[]" />. ';
      $p='Père : ';
      $host="localhost"; //le SGBD se trouve sur la même adresse que le serveur d'application
      $login="root"; //Nom d'utilisateur du SGBD
      $psswd=""; //Mot de passé du SGBD. Ici, il n'y a pas de mot de passe.
      $bdd="exemple_alias"; // Dans la base de données ciblée par les requêtes
      $c=mysqli_connect($host, $login, $psswd, $bdd); //Connexion
      if (!$c)
         die("échec : " . mysqli_connect_error()."</body></html>"); //En cas d'échec quitter
      $q="SELECT `id`,`prenom` FROM `personne`";
      //Envoyer la requête, si problème quitter
      $r=mysqli_query($c,$q) or die("échec : ". mysqli_error($c)."</body></html>");
      if (mysqli_num_rows($r)>0) { //S'il y a des resultants (table personne non vide)
         $p.='<select name="pere[]">';
         while($l=mysqli_fetch_assoc($r))
            //Afficher les prénoms de chaque personne dans les options du select
            $p.='<option value="' . $l["id"] . '">' . $l["prenom"] . "</option>";
         mysqli_free_result($r); //Libérer l'espace alloué aux resultants envoys par le SGBD
         $p.='</select>.';
      }
      else {
         die("échec : Aucun père"."</body></html>");
      }
      mysqli_close($c); //Se déconnecter du SGBD
```

```
    ?>
    <h1>Insertion de trois personnes :</h1>
    <form method="post" action="r1.php">
      <?php //Afficher le formulaire s'il n'y pas d'erreurs.
        for($i=0;$i<3;$i++) {
          echo $n . $f . $p . "<br />";
        }
      ?>
      <input type="submit" value="Enregistrer" />
    </form>
  </body>
  </html>
```

Script *r.php*

```
  <html>
  <head>
    <meta charset="utf-8" />
    <title>Exemple de BdD : MySQLi Procédural</title>
  </head>
  <body>
    <?php
      $host="localhost";
      $login="root";
      $psswd="";
      $bdd="exemple_alias";
      $c=mysqli_connect($host, $login, $psswd, $bdd);
      if (!$c) die("échec : " . mysqli_connect_error()."</body></html>");
      for ($i=0; $i<3; $i++) {
        $q='INSERT INTO `personne` (`nom`,
        `prenom`, `id_pere`) VALUES ("';
        $q.=$_POST['nom'][$i];
        $q.='","';
        $q.=$_POST['prenom'][$i];
        $q.='","';
        $q.=$_POST['pere'][$i];
        $q.='");';
        mysqli_query($c,$q) or die("échec : ". mysqli_error($c)."</body></html>");
      }
      mysqli_close($c);
    ?>
    <p>Insertion réussie</p>
    <p><a href="e1.php">Nouvelle insertion</a></p>
    <p><a href="index.htm">Retour</a></p>
  </body>
  </html>
```

API MySQLi orienté objet

Script *e.php*

```
  <html>
  <head>
    <meta charset="utf-8" />
    <title>Exemple de BdD : MySQLi OO</title>
  </head>
  <body>
    <?php
      $n='Nom : <input type="text" name="nom[]" />. ';
      $f='Prénom : <input type="text" name="prenom[]" />. ';
      $p='Père : ';
      $host="localhost";
      $login="root";
      $psswd="";
      $bdd="exemple_alias";
```

```
        $c=new mysqli($host, $login, $psswd, $bdd);
        if (!$c) die("échec : " . $c->connect_error()."</body></html>");
        $q="SELECT `id`,`prenom` FROM `personne`";
        $r = $c->query($q) or die("échec : ". $c->error."</body></html>");
        if ($r->num_rows>0) {
           $p.='<select name="pere[]">';
           while($l=$r->fetch_assoc())
              $p.='<option value="' . $l["id"] . '">' . $l["prenom"] . "</option>";
           $r->close();
           $p.='</select>.';
        }
        else {
           die("échec : Aucun père"."</body></html>");
        }
        $c->close();
     ?>
     <h1>Insertion de trois personnes :</h1>
     <form method="post" action="r2.php">
        <?php
           for($i=0;$i<3;$i++) {
              echo $n . $f . $p . "<br />";
           }
        ?>
        <input type="submit" value="Enregistrer" />
     </form>
  </body>
  </html>
```

Script *r.php*

```
  <html>
  <head>
     <meta charset="utf-8" />
     <title>Exemple de BdD : MySQLi OO</title>
  </head>
  <body>
     <?php
        $n='Nom : <input type="text" name="nom[]" />. ';
        $f='Prénom : <input type="text" name="prenom[]" />. ';
        $p='Père : ';
        $host="localhost";
        $login="root";
        $psswd="";
        $bdd="exemple_alias";
        $c=new mysqli($host, $login, $psswd, $bdd);
        if (!$c) die("échec : " . $c->connect_error()."</body></html>");
        for ($i=0; $i<3; $i++) {
           $q='INSERT INTO `personne` (`nom`,`prenom`, `id_pere`) VALUES ("';
           $q.=$_POST['nom'][$i];
           $q.='","';
           $q.=$_POST['prenom'][$i];
           $q.='","';
           $q.=$_POST['pere'][$i];
           $q.='");';
           $c->query($q) or die("échec : ". $c->error."</body></html>");
        }
        $c->close();
     ?>
     <p>Insertion réussie</p>
     <p><a href="e2.php">Nouvelle insertion</a></p>
     <p><a href="index.htm">Retour</a></p>
  </body>
  </html>
```

API PDO

Script *e.php*

```
<html>
<head>
   <meta charset="utf-8" />
   <title>Exemple de BdD : PDO</title>
</head>
<body>
   <?php
      $n='Nom : <input type="text" name="nom[]" />. ';
      $f='Prénom : <input type="text" name="prenom[]" />. ';
      $p='Père : ';
      $host="localhost";
      $login="root";
      $psswd="";
      $bdd="exemple_alias";
      $c=new PDO("mysql:host=$host;dbname=$bdd", $login, $psswd);
      if (!$c) die("échec : " . $c->errorInfo()."</body></html>");
      $q="SELECT `id`,`prenom` FROM `personne`";
      $s = $c->query($q) or die("échec : ". $s->errorInfo."</body></html>");
      $s->setFetchMode(PDO::FETCH_OBJ);
      $p.='<select name="pere[]">';
      while($l=$s->fetch())
      $p.='<option value="' . $l->id . '">' .
      $l->prenom . "</option>";
      $p.='</select>.';
      $l=NULL;
      $c=NULL;
   ?>
   <h1>Insertion de trois personnes :</h1>
   <form method="post" action="r3.php">
      <?php
         for($i=0;$i<3;$i++) {
            echo $n . $f . $p . "<br />";
         }
      ?>
      <input type="submit" value="Enregistrer" />
   </form>
</body>
</html>
```

Script *r.php*

```
<html>
<head>
   <meta charset="utf-8" />
   <title>Exemple de BdD : PDO</title>
</head>
<body>
   <?php
      $n='Nom : <input type="text" name="nom[]" />. ';
      $f='Prénom : <input type="text" name="prenom[]" />. ';
      $p='Père : ';
      $host="localhost";
      $login="root";
      $psswd="";
      $bdd="exemple_alias";
      $c=new PDO("mysql:host=$host;dbname=$bdd", $login, $psswd);
      if (!$c) die("échec : " . $c->errorInfo()."</body></html>");
      $q='INSERT INTO `personne` (`nom`,`prenom`,
      `id_pere`) VALUES (:nom,:prenom,:pere);';
      $s = $c->prepare($q);
      $s-> bindparam (':nom', $n);
```

```
        $s-> bindparam (':prenom', $f);
        $s-> bindparam (':pere', $p);
        for ($i=0; $i<3; $i++) {
           $n=$_POST['nom'][$i];
           $f=$_POST['prenom'][$i];
           $p=$_POST['pere'][$i];
           $s-> execute();
        }
        $c=NULL;
     ?>
     <p>Insertion réussie</p>
     <p><a href="e1.php">Nouvelle insertion</a></p>
     <p><a href="index.htm">Retour</a></p>
  </body>
 </html>
```