

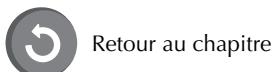
MySQL, mode console et jointures

Sommaire

I. Accéder au mode console avec Wamp	3
II. Création de la base de données	7
A. Création des tables.....	7
B. Les requêtes de jointures.....	10
C. Les alias.....	10
D. INNER JOIN, jointure interne	11
E. LEFT OUTER JOIN et RIGHT OUTER JOIN, les jointures externes	12
F. Les jointures sur plusieurs tables.....	13

Crédits des illustrations:
© DR.

Les repères de lecture



Retour au chapitre



Définition



Objectif(s)



Espace Élèves



Vidéo /
Audio



Point important /
À retenir



Remarque(s)



Pour aller
plus loin



Normes et lois



Quiz

Nous avons abordé, dans le module précédent, les bases de données MySQL en utilisant l'utilitaire PhpMyAdmin qui est disponible dans WAMP.

Il est utile de s'intéresser à l'usage de MySQL en mode console afin de s'affranchir de PhpMyAdmin ou des autres utilitaires en mode graphique qui peuvent ne pas être disponibles dans le cadre d'une intervention ou d'un projet.

Par ailleurs, l'usage du mode console permet de bien travailler ses requêtes et ainsi d'améliorer ses connaissances. Nous noterons par ailleurs que les performances de MySQL sont meilleures en mode console en offrant une immédiateté de résultat car le traitement PHP de PhpMyAdmin n'est de fait pas utilisé.

I. Accéder au mode console avec Wamp

Le mode console de MySQL s'active depuis le menu de PhpMyAdmin, que nous connaissons bien, via l'élément de liste intitulé « MySQL » qui ouvre le sous-menu contenant l'élément de liste « Console MySQL ». Il suffit de cliquer sur cet élément pour ouvrir la console.

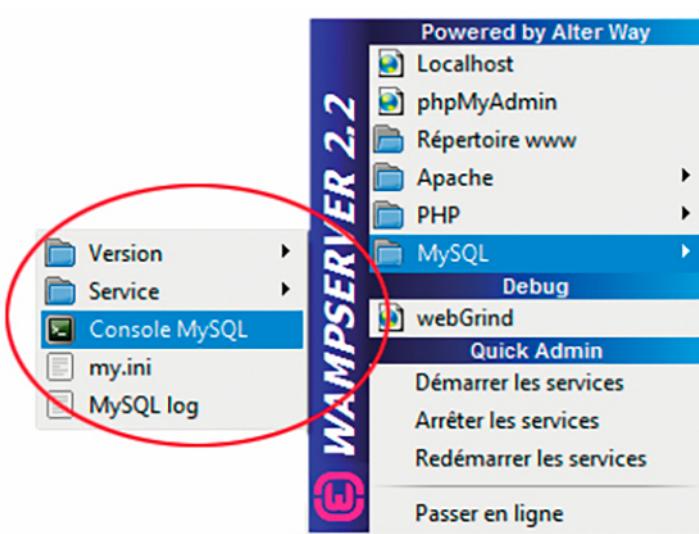


Fig. 1 Accès au mode console de MySQL

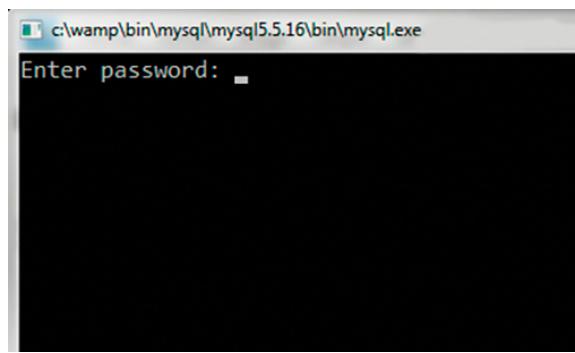


Fig.2 La console MySQL

Avant de commencer son utilisation, il peut être utile, si souhaité, de procéder à une légère configuration simple consistant à obtenir le mode console en plein écran. Par défaut, celle-ci occupe en effet un quart d'écran et n'est pas une fenêtre directement redimensionnable par la souris.

Il faut cliquer avec le bouton de droite sur le haut de la fenêtre (encadré en rouge) puis cliquer sur *Propriétés*.

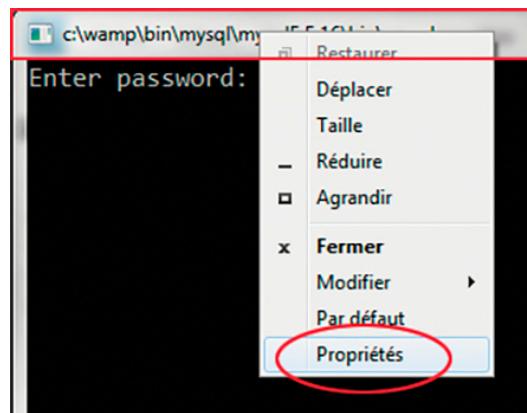


Fig.3 Accès aux propriétés

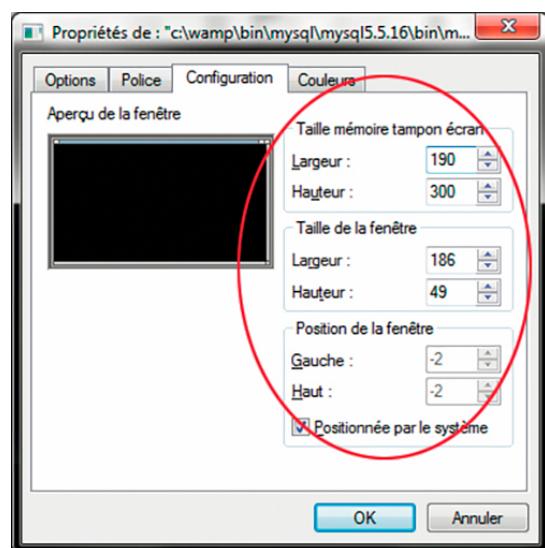


Fig.4 Modifier la taille de la fenêtre

Dans le menu proposé, il est possible de configurer la taille de la fenêtre selon ses souhaits (ici largeur: 186 et hauteur: 46). Il est également possible de modifier les couleurs, la police, la taille du curseur, etc.

Ceci étant effectué, nous pouvons utiliser la console.

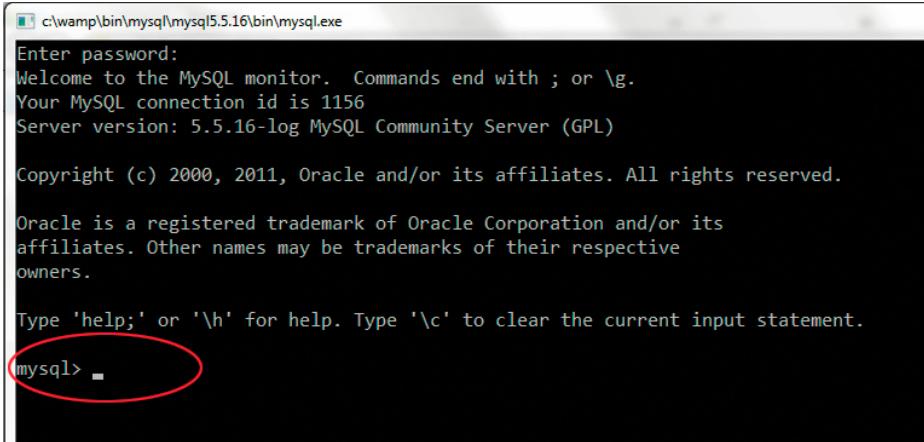
Comme nous le constatons, nous sommes invités à entrer un mot de passe (*Enter password*).

Ce mot de passe est celui du serveur de la base de données MySQL. Si nous utilisons WAMP, et à moins d'en avoir spécifié une autre, la valeur du mot de passe par défaut est vide (et le *login* est *root*, c'est d'ailleurs bien ce que nous indiquons dans les fichiers PHP dédiés à la connexion avec la base de données que nous utilisons).

Puisque le mot de passe est vide, nous validons directement au clavier avec la touche « Entrée » (Enter), sinon nous saisissons le mot de passe avant de valider.

À ce moment, un message de bienvenue apparaît contenant quelques informations.

En dessous nous remarquons l'invite **mysql >** qui sera en permanence à notre disposition et tout ce que nous écrirons sera visible en face de cette invite.



```
c:\wamp\bin\mysql\mysql5.5.16\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1156
Server version: 5.5.16-log MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> -
```

Fig.5 Message de bienvenue et invite MySQL (entouré en rouge)

C'est donc face à cette invite que nous pourrons écrire nos requêtes MySQL. La première requête sera celle destinée à lister toutes les bases de données existantes afin de choisir laquelle nous allons utiliser pour travailler.

La requête permettant de lister les bases de données est:

```
SHOW DATABASES;
```

Fig.6

Nous écrivons donc cette requête puis validons avec « Entrée ». MySQL retourne alors le résultat de la requête en affichant l'ensemble des bases de données du serveur.

```

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| test |
+-----+
4 rows in set (0.03 sec)

```

Fig.7

Affichage des bases de données (ici les bases par défaut de MySQL).

Avant d'aller plus loin, voici quelques informations utiles concernant la console.

Le point-virgule

Il faut toujours ajouter un point-virgule ; à la fin de la requête avant de la valider avec Entrée. Sinon, MySQL considère à juste titre que la requête n'est pas terminée et va seulement aller à la ligne.

Le copier-coller

Le copier-coller ne fonctionne pas directement avec les raccourcis clavier.

Pour coller du code dans la console, il est nécessaire de cliquer avec le bouton droit de la souris et choisir « coller ».

Pour copier du code dans le presse-papiers depuis la console, il faut, via le bouton droit de la souris, cliquer sur l'option « sélectionner » puis sélectionner avec le bouton de gauche et enfin effectuer un copier avec le raccourci clavier. Cette méthode n'est pas très simple (et ne marche pas toujours très bien) et nous verrons comment récupérer le code dans un fichier texte autrement.

La touche F3

La touche F3 est très pratique car elle permet de répéter la dernière ligne de la requête précédente.

Enfin, **il est très important de bien surveiller quelles requêtes vous validez car MySQL ne vous demandera pas de confirmation, contrairement à PhpMyAdmin !** Si vous validez une requête de suppression de données ou de destruction d'une base de données, MySQL détruira simplement et définitivement les données ou la base de données.

Ces précautions dites, nous pouvons travailler sur une base de données. Pour travailler sur une base, il faut indiquer à MySQL quelle base nous voulons utiliser au moyen de USE:

```
USE nom_de_la_base;
```

Fig.8

Dans le cadre de ce module, nous allons créer une base de données simple contenant deux tables. La base est nommée *dbtest* et contient les tables *auteur* et *livre*, qui contiendront respectivement quatre auteurs et six livres.



Attention ! Pas de message de confirmation en mode console !

Pour les besoins de l'exercice, nous considérerons qu'un seul auteur peut écrire plusieurs livres et nous obtiendrons la structure suivante avec une référence de l'identifiant de l'auteur dans la table livre.

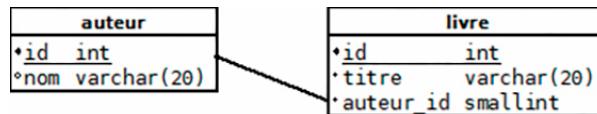


Fig.9 Structure des tables

II. Création de la base de données

Voici le code MySQL (vous pouvez utiliser PhpMyAdmin)

```
CREATE DATABASE dbtest CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Fig.10

Remarquons que nous demandons **lors de sa création** que **la base de données soit encodée en utf8**, ce qui permettra que toutes les tables que nous créerons par la suite soient en utf8 même si nous ne le précisons pas. **L'encodage utf8 permet d'inclure tout type de contenu accentué, c'est une norme qu'il est nécessaire d'utiliser pour tous nos développements.** L'option d'encodage (collation) qui est choisie est *utf8_general_ci*, les lettres *ci* signifiant *case insensitive*, ce qui signifie donc insensible à la casse ce qui permet donc de récupérer des résultats pour une recherche par exemple de *livre* ou de *LIVRE*.

A. Création des tables

```
CREATE TABLE IF NOT EXISTS `auteur` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `nom` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=5 ;

-- Contenu de la table `auteur`

INSERT INTO `auteur` (`id`, `nom`) VALUES
(1, 'Claude'),
(2, 'Marie'),
(3, 'Pierre'),
(4, 'Jean');

-- Structure de la table `livre`

CREATE TABLE IF NOT EXISTS `livre` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `titre` varchar(20) DEFAULT NULL,
  `auteur_id` smallint(10) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=7 ;

-- Contenu de la table `livre`

INSERT INTO `livre` (`id`, `titre`, `auteur_id`) VALUES
(1, 'PHP et Mysql', 2),
(2, 'Apprendre PHP', 2),
(3, 'Cours de SQL', 3),
(4, 'PHP par la pratique', 4),
(5, 'Projet PHP', 2),
(6, 'PHP et Ajax', 3);
```

Fig.11

Une fois la base et les tables créées, **il faut indiquer à MySQL que nous souhaitons utiliser cette base avec USE**:

```
USE dbtest;
```

Fig. 12

MySQL répond: **Database changed**. Ce qui signifie que nous travaillons bien désormais sur cette base de données pour laquelle nous pouvons commencer par lister les tables de la base avec l'instruction SHOW TABLES.

```
SHOW TABLES;
```

Tables_in_dbtest
auteur
livre

2 rows in set (0.00 sec)

Fig. 13

Comme nous le voyons, MySQL nous restitue alors la liste des tables sous la forme visuelle d'un tableau intitulé *Tables_in_dbtest*.

Le nombre de résultat est fourni : *2 rows in set* ainsi que le délai pris pour traiter la requête et répondre (0 seconde 0 dixième et 0 centième).

Il est utile de demander à accéder à la structure des tables (comme avec PhpMyAdmin) avec l'instruction **DESCRIBE**.



L'invite mysql> est celle de la console.

```
DESCRIBE auteur;
```

```
mysql> DESCRIBE auteur;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
nom	varchar(10)	YES		NULL	

2 rows in set (0.08 sec)

```
mysql> DESCRIBE livre;
```

Field	Type	Null	Key	Default	Extra
id	int(10) unsigned	NO	PRI	NULL	auto_increment
titre	varchar(20)	YES		NULL	
auteur_id	smallint(10) unsigned	NO		NULL	

3 rows in set (0.08 sec)

Fig. 14

Enfin nous pouvons effectuer un **SELECT** permettant d'afficher les données.

```
mysql> SELECT * FROM auteur;
```

id	nom
1	Claude
2	Marie
3	Pierre
4	Jean

4 rows in set (0.00 sec)

Fig.15

```
mysql> SELECT * FROM livre;
```

id	titre	auteur_id
1	PHP et Mysql	2
2	Apprendre PHP	2
3	Cours de SQL	3
4	PHP par la pratique	4
5	Projet PHP	2
6	PHP et Ajax	3

6 rows in set (0.00 sec)

Fig.16

Pour mémoire, le signe * qui permet de récupérer tous les contenus est très consommateur de ressources et s'il peut être utilisé en mode console ou dans les tests de nos scripts, son usage est à proscrire pour un site en production, accessible aux utilisateurs.

Il est très pratique de récupérer les résultats obtenus dans un fichier texte et pour ce faire nous utilisons l'instruction **TEE** comme ceci :

```
TEE chemin_ver_le_fichier;
```

Fig.17

Par défaut, MySQL crée le fichier souhaité s'il n'existe pas et le chemin est **C:\Wamp**, ce qui signifie que le fichier sera créé dans ce répertoire, par exemple **fichier.txt** si nous écrivons cette requête :

```
TEE fichier.txt; -- MySQL se connecte au fichier
```

Fig.18

Si nous souhaitons enregistrer les résultats dans un fichier dont le chemin est différent, il faut doubler les anti-slash comme ceci :

```
TEE C:\\wamp\\\\www\\\\dir\\\\fichier.txt;
```

Fig.19

MySQL répond alors : Logging to file **fichier.txt** et l'ensemble des requêtes saisies et des résultats retournés seront donc écrits dans ce fichier. Si nous souhaitons arrêter cette écriture dans le fichier il faut utiliser l'instruction **NOTEES**; ce à quoi MySQL répond *Outfile disabled*.

```
NOTEES ; -- MySQL se déconnecte du fichier en cours
```

Fig.20

Après avoir vu le principe d'usage de la console MySQL, nous pouvons aborder les requêtes plus complexes et notamment les requêtes de jointure.

B. Les requêtes de jointures

Les requêtes de jointure sont utilisées dès que nous avons une relation entre les données de deux tables et que nous souhaitons en rapprocher les résultats. Une jointure est en effet un rapprochement, une fusion et dans ce cas, **joindre** signifie alors **relier**.

Dans notre exemple, la table **livre** contient l'identifiant de l'auteur qui a écrit le livre et une première requête de jointure consistera alors à afficher les livres écrits par un auteur donné.

Si nous cherchons les livres dont l'auteur est Marie, il nous faudrait effectuer deux requêtes :

- **Une requête pour déterminer l'identifiant de l'auteur**

```
SELECT id FROM auteur WHERE nom LIKE '%marie%'; -- résultat : 2
```

Fig.21

- **Une requête pour déterminer quels sont les livres dont l'auteur a pour identifiant la valeur 2.**

```
SELECT titre FROM livre WHERE auteur_id = 2;  
-- résultat : Apprendre PHP, PHP et Mysql, Projet PHP
```

Fig.22

Avec les jointures, nous pourrons effectuer cette recherche en une seule requête.

C. Les alias

Un alias permet de renommer une table durant une requête, ce qui permet de distinguer les résultats, notamment pour deux tables ayant chacune une colonne *id*.

Soit, par exemple, la requête suivante destinée à afficher toutes les valeurs des colonnes *id*. des deux tables *auteur* et *livre* :

```
SELECT id, id FROM auteur, livre;
```

Fig.23

MySQL ne sait alors pas à quelle colonne associer les résultats car elles ont le même nom et sa réponse est alors:

```
ERROR 1052 (23000): Column 'id' in field list is ambiguous
```

Fig.24

Les alias vont nous permettre de bien distinguer les colonnes en spécifiant que A est l'alias de la table auteur et que L est l'alias de la table livre:

```
SELECT A.id, L.id FROM auteur AS A, livre AS L;
```

Fig.25

A.id fait donc bien référence à la colonne de la table auteur dont l'alias est A tandis que L.id fait bien référence à la colonne de la table livre dont l'alias est L.

Même si les noms des colonnes ne sont pas identiques, utiliser un alias est toujours une bonne idée afin de bien structurer nos requêtes. À noter que le mot-clé AS est facultatif, il est cependant fortement recommandé de toujours l'employer.

D. INNER JOIN, jointure interne

Pour travailler sur ces deux tables, nous utilisons comme ceci la jointure interne **INNER JOIN** afin d'obtenir le nom de l'auteur et les titres de ses livres:

Tableau n°1

SELECT * FROM auteur AS A	Obtenir les valeurs depuis la table auteur (pour laquelle nous utiliserons l'alias A).
INNER JOIN livre AS L	En la joignant (fusionnant) avec la table livre (pour laquelle nous utiliserons l'alias L) dont nous récupérerons aussi les valeurs.
ON A.id. = L.auteur_id.	Et en effectuant cette jointure sur le fait que l'identifiant de la table auteur (A étant l'alias de la table auteur) est le même que auteur_id. de la table livre (L étant l'alias de la table livre).
WHERE A.id. = 2	Lorsque l'identifiant de l'auteur a comme valeur 2.

Nous voyons que la jointure **INNER JOIN** s'exprime avec l'opérateur **ON** qui permet de spécifier les valeurs que nous allons utiliser: la colonne id. de la table auteur et la colonne auteur_id. de la table livre, chacune de ces colonnes devant contenir les mêmes valeurs.

```
mysql> SELECT * FROM auteur AS A
-> INNER JOIN livre AS L
-> ON A.id = L.auteur_id
-> WHERE A.id = 2;
```

Fig.26

Le résultat obtenu est le suivant:

id	nom	id	titre	auteur_id
2	Marie	2	Apprendre PHP	2
2	Marie	1	PHP et Mysql	2
2	Marie	5	Projet PHP	2

Fig.27

Nous constatons que ce résultat est bien constitué par la jointure:

- des deux colonnes de la table *auteur*;
- et des trois colonnes de la table *livre*.

Nous pouvons également effectuer une requête plus générale et obtenir la liste des auteurs de chaque livre, nous ne spécifierons donc pas de clause **WHERE**.

```
mysql> SELECT * FROM auteur INNER JOIN livre ON auteur.id = livre.auteur_id;
```

id	nom	id	titre	auteur_id
2	Marie	1	PHP et Mysql	2
2	Marie	2	Apprendre PHP	2
2	Marie	5	Projet PHP	2
3	Pierre	3	Cours de SQL	3
3	Pierre	6	PHP et Ajax	3
4	Jean	4	PHP par la pratique	4

Fig.28

La jointure **INNER JOIN** relie donc les deux tables tout en étant obligée d'obtenir des données pour chaque table. Il n'y a donc pas de valeurs vides (**NULL**) dans le résultat. **L'obtention de valeurs vides sera possible uniquement avec les jointures externes.**

E. LEFT OUTER JOIN et RIGHT OUTER JOIN, les jointures externes

Les jointures externes offrent donc la possibilité de récupérer des valeurs vides.

Si un auteur enregistré dans la table n'a pas écrit de livre et que donc il n'existe pas de correspondance dans la table *livre* pour cet auteur, une jointure interne ne restituera pas cet auteur tandis qu'une jointure externe le restituera.

Une jointure externe est constituée par la commande **OUTER JOIN** que l'on préfixe avec **LEFT** ou **RIGHT** pour spécifier si nous souhaitons toutes les lignes de la table qui est mentionnée à gauche ou à droite dans l'écriture de la requête.



LEFT OUTER JOIN peut s'écrire **LEFT JOIN**.

RIGHT OUTER JOIN peut s'écrire **RIGHT JOIN**.

Exemples

Tableau n°2

SELECT * FROM auteur LEFT JOIN livre ON auteur.id = livre.auteur_id; ;	La table <i>auteur</i> est à gauche de LEFT JOIN, et puisque nous utilisons LEFT alors nous récupérons toutes les lignes de la table <i>auteur</i> même si elles n'ont pas toutes des correspondances dans la table <i>livre</i> .
SELECT * FROM auteur RIGHT JOIN livre ON auteur.id = livre.auteur_id; ;	La table <i>livre</i> est à droite de RIGHT JOIN, et puisque nous utilisons RIGHT alors nous récupérons toutes les lignes de la table <i>livre</i> même si elles n'ont pas toutes des correspondances dans la table <i>auteur</i> .

Résultat pour la première requête:

```
mysql> SELECT * FROM auteur LEFT JOIN livre ON auteur.id = livre.auteur_id;
+---+-----+---+-----+-----+
| id | nom  | id | titre | auteur_id |
+---+-----+---+-----+-----+
| 1  | Claude | NULL | NULL | NULL      |
| 2  | Marie   | 1    | PHP et Mysql | 2          |
| 2  | Marie   | 2    | Apprendre PHP | 2          |
| 2  | Marie   | 5    | Projet PHP | 2          |
| 3  | Pierre  | 3    | Cours de SQL | 3          |
| 3  | Pierre  | 6    | PHP et Ajax | 3          |
| 4  | Jean    | 4    | PHP par la pratique | 4          |
+---+-----+---+-----+-----+
7 rows in set (0.07 sec)
```

Fig.29

Comme nous le voyons, nous récupérons l'auteur Claude qui n'a écrit aucun livre et le résultat correspondant dans la table *livre*. Ce résultat retourné est **NULL**, c'est-à-dire qu'il n'existe pas de valeur correspondante (comme nous l'avons déjà vu, **NULL** est l'absence de valeurs).

Le résultat serait identique pour la requête **RIGHT JOIN** qui intervertirait les noms des tables (la table *auteur* est cette fois à droite), même si dans ce cas les colonnes de la table *livre* sont retournées en premier:

```
mysql> SELECT * FROM livre RIGHT JOIN auteur ON auteur.id = livre.auteur_id;
+---+-----+-----+---+-----+
| id | titre | auteur_id | id | nom  |
+---+-----+-----+---+-----+
| NULL | NULL | NULL | 1 | Claude |
| 1 | PHP et Mysql | 2 | 2 | Marie |
| 2 | Apprendre PHP | 2 | 2 | Marie |
| 5 | Projet PHP | 2 | 2 | Marie |
| 3 | Cours de SQL | 3 | 3 | Pierre |
| 6 | PHP et Ajax | 3 | 3 | Pierre |
| 4 | PHP par la pratique | 4 | 4 | Jean |
+---+-----+-----+---+-----+
7 rows in set (0.01 sec)
```

Fig.30

F. Les jointures sur plusieurs tables

Il est possible d'effectuer des jointures sur plusieurs tables pour établir des relations entre des données qui ne sont pas directement stipulées, comme, un auteur et un thème.

Nous ajoutons à la base de données la table *theme* qui pourra établir le thème d'un des ouvrages de la base et qui contient quatre lignes.

```
CREATE TABLE IF NOT EXISTS `theme` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=5;

INSERT INTO `theme` (`id`, `theme`) VALUES
(1, 'PHP'), (2, 'MySQL'), (3, 'Ajax'), (4, 'HTML');
```

Fig.31



Il ne faut pas confondre **les jointures internes ou externes** qui sont des requêtes associant les données avec **les tables associatives** qui permettent de stocker une relation entre les enregistrements de deux tables.

Cette fois, la règle que nous édictons est qu'un livre peut être associé à plusieurs thèmes, par exemple, le livre « PHP et MySQL » a deux thèmes : PHP et MySQL. Puisque nous ne pouvons donc pas utiliser de colonne *livre_id*. dans la table *theme* nous allons créer une table *theme_livre* qui nous servira de table associant les identifiants (on parle alors de table associative).

```
CREATE TABLE IF NOT EXISTS `theme_livre` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `livre_id` int(11) NOT NULL,
  `theme_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=9;

INSERT INTO `theme_livre` (`id`, `livre_id`, `theme_id`) VALUES
(1, 1, 1), (2, 2, 1), (3, 4, 1), (4, 5, 1), (5, 6, 1), (6, 1, 2),
(7, 3, 1), (8, 6, 3);
```

Fig.32



Pour une table associative, définir une clef primaire est toujours une bonne pratique.

Cette nouvelle table permet d'associer un ou plusieurs thèmes à un livre, par exemple le livre « PHP et Ajax » dont l'identifiant *id*. est 6 est ainsi associé aux thèmes PHP (*theme_id*. = 1) et Ajax (*theme_id*. = 3).

Voici le schéma de la base de données :



Fig.33 Structure de la base de données

Comme nous le voyons la table associative *theme_livre* permet de relier par leurs identifiants respectifs *livre_id*. et *theme_id*. les données de la table *livre* et celles de la table *theme*.

Pour récupérer les livres et les thèmes correspondants, nous procédons à une requête contenant une jointure externe :

```
mysql> SELECT L.titre, T.nom FROM livre AS L
-> INNER JOIN theme_livre AS J
-> ON L.id = J.livre_id
-> INNER JOIN theme AS T
-> ON T.id = J.theme_id;
```

Fig.34

titre	nom
PHP et Mysql	PHP
Apprendre PHP	PHP
PHP par la pratique	PHP
Projet PHP	PHP
PHP et Ajax	PHP
Cours de SQL	PHP
PHP et Mysql	MySQL
PHP et Ajax	Ajax

Fig.35

Si nous souhaitons obtenir pour chaque auteur les thèmes abordés dans ses livres, nous pouvons effectuer une **succession de trois jointures** (en nous servant du schéma précédent) :

```
mysql> SELECT A.nom, T.nom
-> FROM auteur AS A
-> INNER JOIN livre AS L
-> ON A.id = L.auteur_id
-> INNER JOIN theme_livre AS J
-> ON L.id = J.livre_id
-> INNER JOIN theme AS T
-> ON T.id = J.theme_id;
```

Fig.36

Nous établissons une jointure entre la table auteur et la table livre, afin de définir la relation *auteur/livre*, puis entre la table livre et la table associative *theme_livre* puis entre la table associative *theme_livre* et la table *theme*, afin de définir la relation *livre/theme*. Nous obtenons ainsi la relation entre les auteurs et les thèmes.

nom	nom
Marie	PHP
Marie	PHP
Jean	PHP
Marie	PHP
Pierre	PHP
Pierre	PHP
Marie	MySQL
Pierre	Ajax

8 rows in set (0.01 sec)

Fig.37

Nous obtenons huit résultats et constatons que des auteurs sont affichés plusieurs fois en étant associés aux mêmes thèmes.

Améliorons notre requête afin d'en supprimer les doublons de résultat avec le mot-clé **DISTINCT** et en ordonnant les résultats par ordres alphabétiques avec une clause **ORDER BY**. Nous spécifions également un alias mais cette fois pour les colonnes afin d'obtenir des intitulés plus clairs (*nom auteur* et *nom theme*).



Nous utilisons ici les quotes car ces alias contiennent des espaces.

```
mysql> SELECT DISTINCT A.nom AS 'nom auteur', T.nom AS 'nom theme'
-> FROM auteur AS A
-> INNER JOIN livre AS L
-> ON A.id = L.auteur_id
-> INNER JOIN theme_livre AS J
-> ON L.id = J.livre_id
-> INNER JOIN theme AS T
-> ON T.id = J.theme_id
-> ORDER BY A.nom, T.nom;
```

nom auteur	nom theme
Jean	PHP
Marie	MySQL
Marie	PHP
Pierre	Ajax
Pierre	PHP

5 rows in set (0.02 sec)

Fig. 38

Cette fois aucun doublon de résultat n'est retourné et les intitulés de colonnes sont tout à fait explicites.

Les mots réservés de MySQL

Comme dans tout langage informatique et comme en PHP, il est à noter que des mots sont réservés pour MySQL et qu'il n'est donc pas permis de les utiliser.

Leur usage par méconnaissance de cette règle peut générer des erreurs et il est recommandé de vérifier si les noms des bases, des tables ou des colonnes ne font pas partie de la liste des mots réservés. La liste des mots réservés par MySQL est disponible ici:

<https://dev.mysql.com/doc/refman/en/keywords.html>