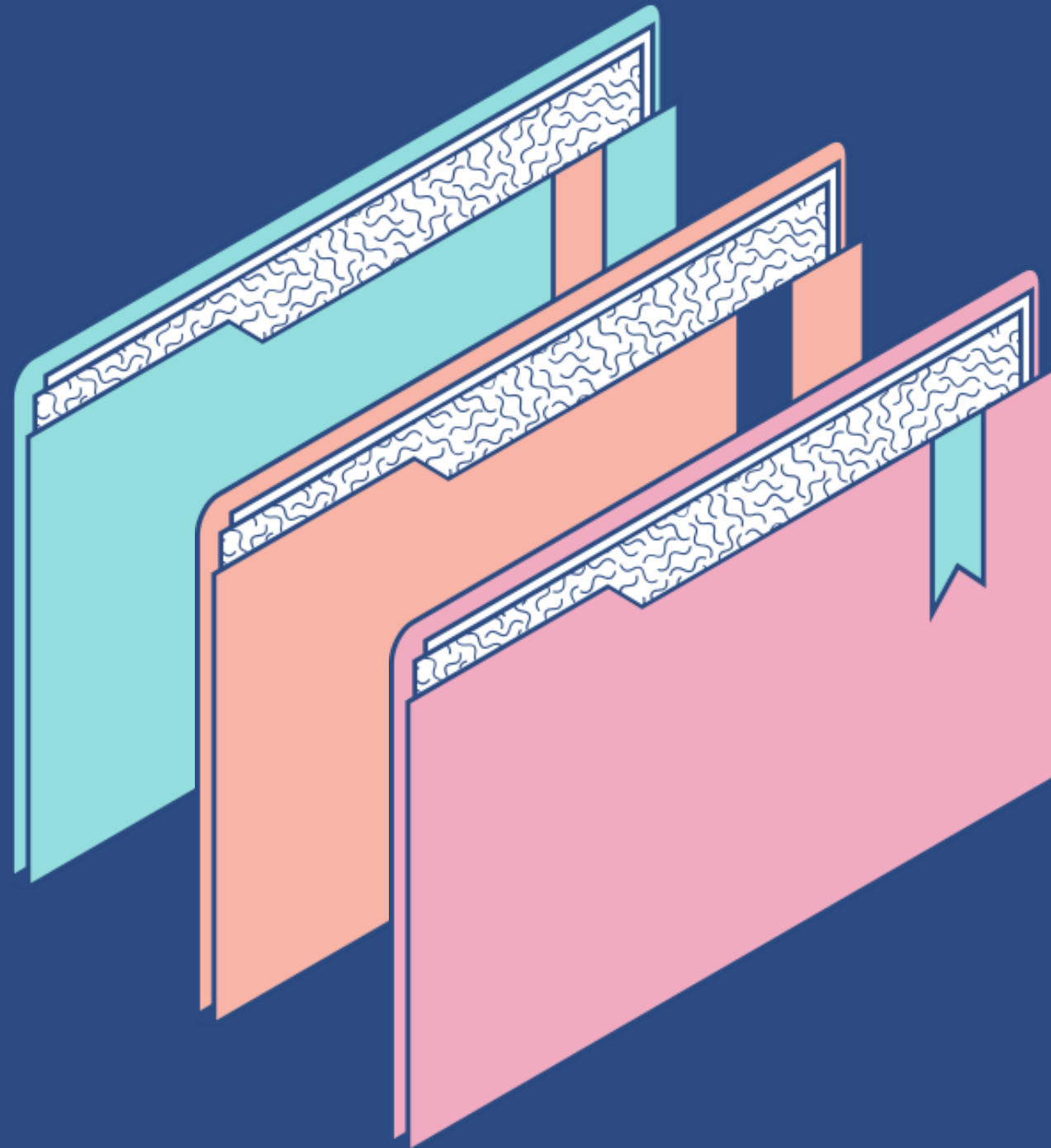


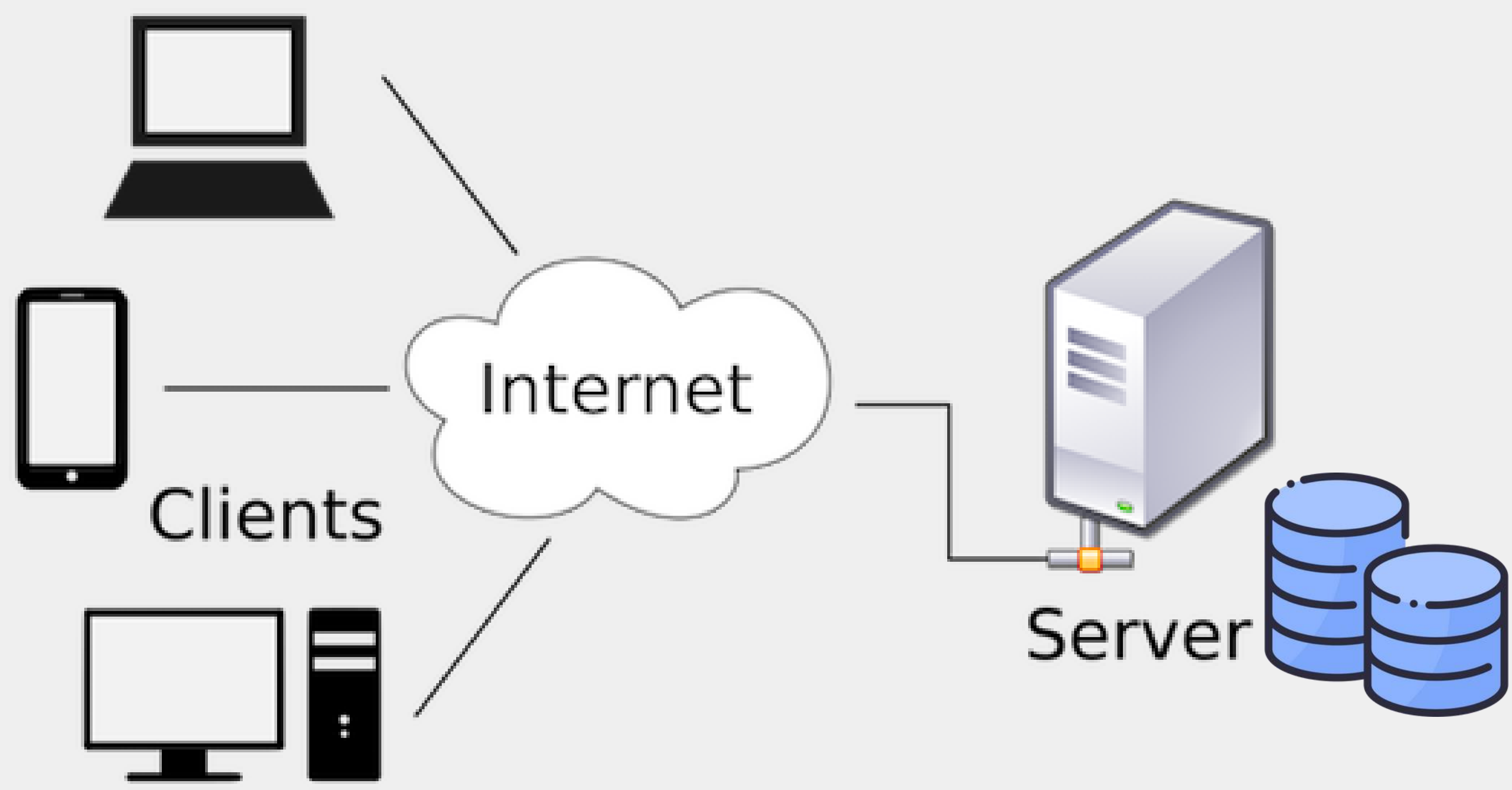
SQL



SOMMAIRE



- Présentation SQL
- Commandes SQL
 - Requêtes de Structure
 - Requêtes de données
- PHPMysqlAdmin





Les outils nécessaires

Pour ce cours, vous aurez besoin de :

- Un serveur (WAMP, XAMPP, LAMP, MAMP, LARAGON ...)
- Un SGBD (PHPMyAdmin, MysqlWorkbench ...)



Utilisation recommandée : XAMPP + PHPMyAdmin



Définitions

SQL : Structured Query Langage

DB/BDD : DataBase/Bases de Données

(R)DBMS/SGBD(R) : (Relational) Database Management System/
Système de Gestion de Bases de Données (Relationnel)



Notions de base

- Champ :
Colonne dans une vue en liste
- Enregistrement ou Tuple :
Ligne dans une vue en liste
- Table:
Contient l'ensemble des enregistrements
- Index(ID) :
Conteneur des clés.

id	nom	prénom	profession	code postal	ville
1	Durand	Michel	Directeur	75016	Paris
2	Dupond	Karine	Secrétaire	92000	Courbevoie
3	Mensoif	Gérard	Commercial	75001	Paris
4	Monauto	Alphonse	Commercial	75002	Paris
5	Emarre	Jean	Employé	75015	Paris
6	Abois	Nicole	Secrétaire	95000	St Denis
7	Dupond	Antoine	Assistant commercial	75014	Paris

	intitulé du champ
	enregistrement
	Champ
	Table



Notions de base

Une base de données est un ensemble qui permet le stockage des données.

Les données sont écrites de manière structurées ce qui signifie que chaque donnée est enregistrée dans un champ qui est inclus dans une table.

Lorsque les tables ont des relations entre elles, on parle alors de bases de données relationnelles.



SGBD(R)

Le SGBD va gérer l'accès à la base de données.

Aucun logiciel n'accédera directement à la base de données, seul le SGBD le fera.

Cela implique que le SGBD :

- Gère le dialogue avec la base de données
- Accède seul à la base de données
- Dispose d'un langage pour pouvoir dialoguer avec les applications, le SQL
- Gère l'écriture et la lecture des données
- Gère le partage des données
- Vérifie l'intégrité des données
- Gère la relation entre les tables, si c'est un SGBDR



Avantages d'une BDD

- Une standardisation des accès :

Chaque logiciel accède aux données via le SGBD en utilisant un langage standardisé. Il est possible de changer de base de données sans avoir à réécrire le logiciel.

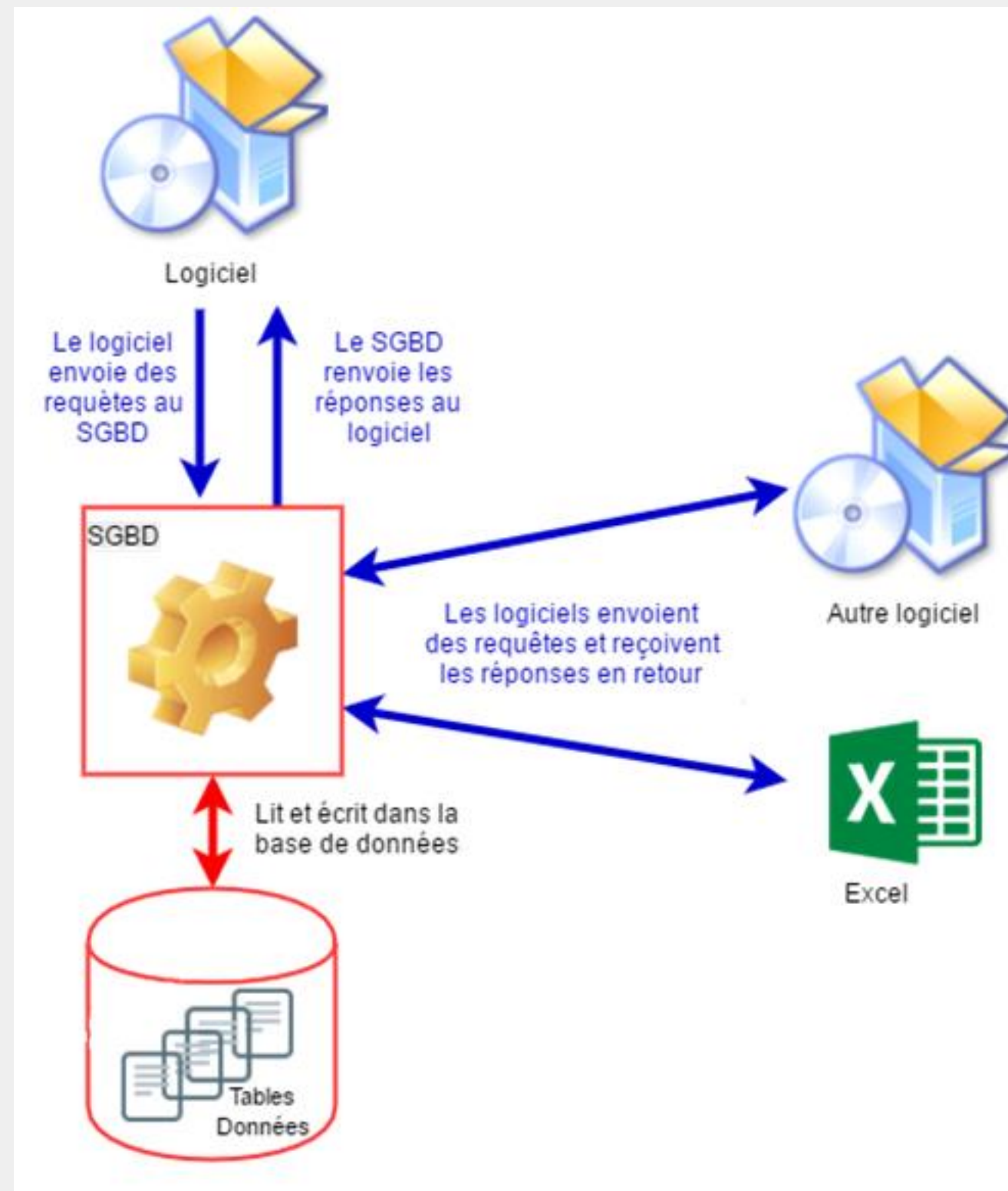
- Un partage des données ainsi que l'accès simultané :

Le SGBD gère les accès à la base de données, donc il gère aussi le partage aux données. Il est donc possible d'avoir plusieurs logiciels qui lisent et écrivent en même temps sur la base de données.

- Une grande fiabilité des données :

Chaque logiciel ne peut pas faire n'importe quoi. C'est le SGBD qui gère l'enregistrement des données.

Avantages d'une BDD





Présentation SQL

Ce langage complet va être utilisé principalement pour:

- Lire les données
- Ecrire les données
- Modifier les données
- Supprimer les données

Il permettra aussi de modifier la structure de la base de données :

- Ajouter des tables
- Modifier les tables
- Supprimer les tables
- Gestion des utilisateurs et de leurs droits
- Gestion des bases de données : en Création, modification...

C.R.U.D
Create
Read
Update
Delete

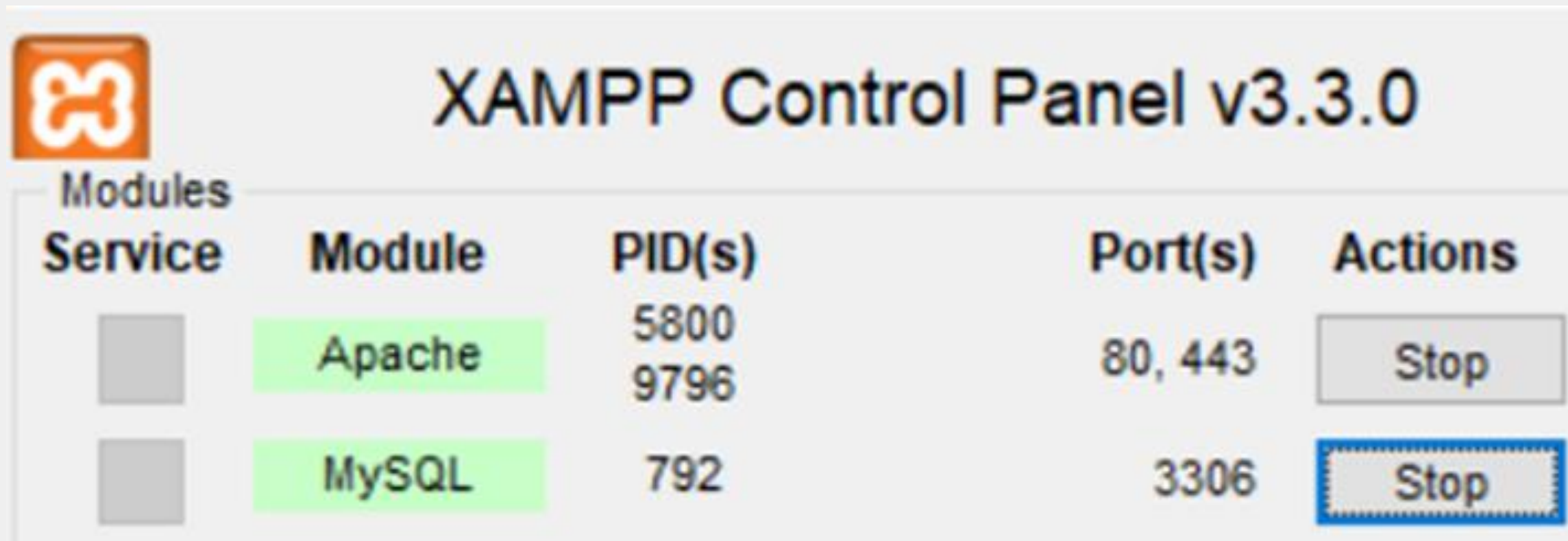
Lancer notre BDD

Lancez votre serveur (WAMP/XAMPP etc...)

Attendez que les voyants passent au vert

Lancez phpMyAdmin en vous rendant à l'adresse suivante:

<http://localhost/phpmyadmin/index.php>



The image shows the XAMPP Control Panel v3.3.0 interface. It features a table with columns for Service, Module, PID(s), Port(s), and Actions. The Apache service is running with PID 5800 and 9796 on ports 80 and 443. The MySQL service is running with PID 792 on port 3306. Both services have a 'Stop' button next to them.

Service	Module	PID(s)	Port(s)	Actions
<input type="checkbox"/>	Apache	5800 9796	80, 443	Stop
<input type="checkbox"/>	MySQL	792	3306	Stop

Commandes SQL

```
4  -- SELECT
5  SELECT *
6  FROM employee
7
8  -- INSERT
9  INSERT INTO employee(emp_id, fname, minit, lname,
10                      job_id, job_lvl, pub_id, hire_date)
11                      VALUES('0000000', 'Almir', 'M', 'Vuk',
12                              7, 12, 1207, 2009-05-09)
13
14  -- UPDATE
15  UPDATE employee
16  SET fname = 'ALMIR'
17  WHERE emp_id = '0000000'
18
19  -- DELETE
20  DELETE
21  FROM employee
22  WHERE emp_id = '0000000'
```



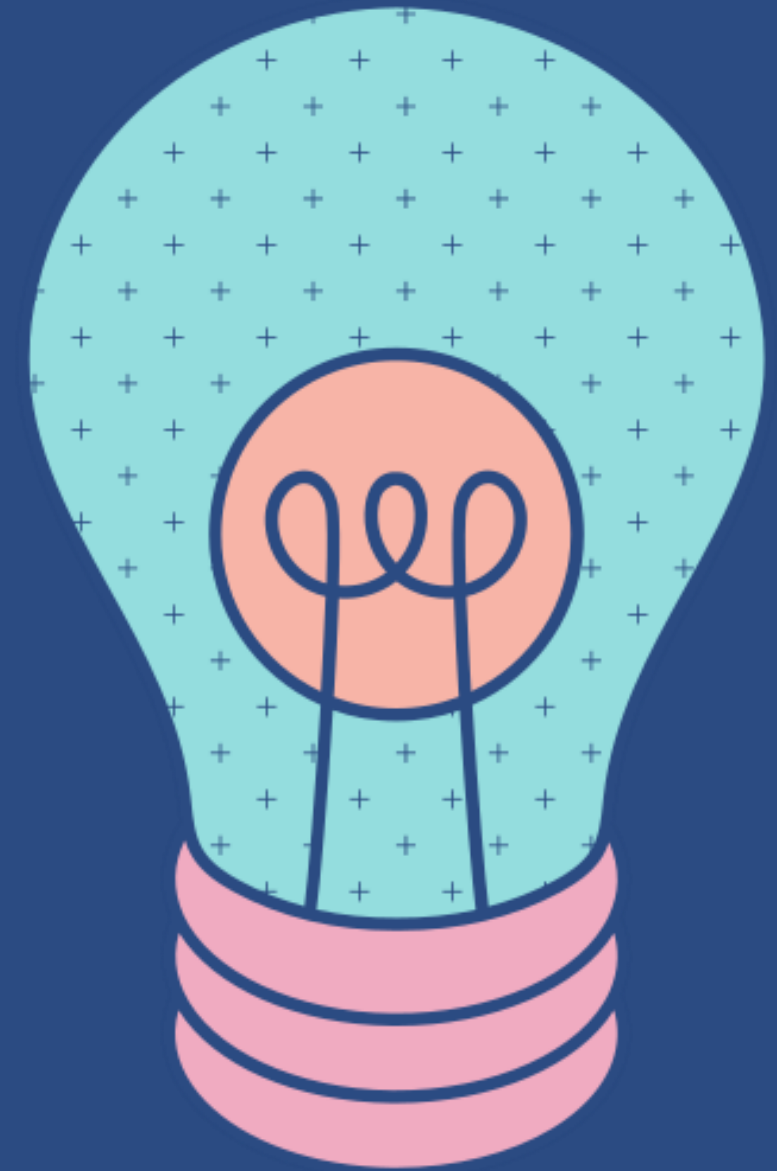
Requêtes de Structure

SQL File 1* x

```
1 • use test;
2
3 • CREATE TABLE employee
4 (
5   id int primary key,
6   name varchar(30),
7   dob datetime,
8   email varchar(40)
9 );
10
11 • DESC employee;
```

Filter: File: Autosize: I A

	Field	Type	Null	Key	Default	Extra
▶	id	int(11)	NO	PRI	NULL	
	name	varchar(30)	YES		NULL	
	dob	datetime	YES		NULL	





Requête CREATE

La requête CREATE permet de :

Créer des bases de données

Créer des tables de données

```
1 CREATE DATABASE test1234;  
2  
3 USE test1234;  
4  
5 CREATE TABLE `users`(  
6     id_user int not null,  
7     name_user varchar(50) not null,  
8     password_user varchar(100) not null,  
9     mail_user varchar(150) not null,  
10    age_user date not null,  
11    id_town int default null  
12 );
```




Requête CREATE

La requête « create database test1234; » permet de créer une base de données nommée « test1234 ».

La requête « use test1234 ; » nous permet de spécifier quelle base de données utiliser lors des prochaines requêtes.

```
1 CREATE DATABASE test1234;
2
3 USE test1234;
4
5 CREATE TABLE `users` (
6     id_user int not null,
7     name_user varchar(50) not null,
8     password_user varchar(100) not null,
9     mail_user varchar(150) not null,
10    age_user date not null,
11    id_town int default null
12 );
```




Requête CREATE

La ligne « CREATE TABLE `users` » permet de créer une table « users » avec à la suite, la liste des colonnes ou des champs qui seront attendus.

ATTENTION: le mot « users » est entouré de « backquote », des apostrophe inversées.

Pour faire des backquotes, il faut faire « alt gr + 7 ».

```
1 CREATE DATABASE test1234;
2
3 USE test1234;
4
5 CREATE TABLE `users` (
6     id_user int not null,
7     name_user varchar(50) not null,
8     password_user varchar(100) not null,
9     mail_user varchar(150) not null,
10    age_user date not null,
11    id_town int default null
12 );
```



Requête CREATE

'id_user', 'name_user', 'password_user'...
seront les noms des champs de notre table.

Le champ 'idtown' correspond également à
la clé primaire d'une autre table.

Nous l'aborderons à nouveau plus tard...

```
1 CREATE DATABASE test1234;
2
3 USE test1234;
4
5 CREATE TABLE `users` (
6     id_user int not null,
7     name_user varchar(50) not null,
8     password_user varchar(100) not null,
9     mail_user varchar(150) not null,
10    age_user date not null,
11    id_town int default null
12 );
```




Requête CREATE

'bigint', 'varchar', 'int' correspondent au type du champ

Il peut être:

Numérique (int , double, real, float)

Chaine de caractère (char, varchar, text, blob...)

Date(date, datetime, timestamp)

Spatial (polygon, point, geometry)

JSON.

(20), (50), (100), etc ... indiquent la taille ou la valeur maximale du champ (non obligatoire pour certains types, comme le 'text' par exemple).

```
1 CREATE DATABASE test1234;
2
3 USE test1234;
4
5 CREATE TABLE `users` (
6     id_user int not null,
7     name_user varchar(50) not null,
8     password_user varchar(100) not null,
9     mail_user varchar(150) not null,
10    age_user date not null,
11    id_town int default null
12 );
```



Requête CREATE

'NOT NULL', 'DEFAULT NULL', signalent à notre SGBD les valeurs par défaut de nos divers champs lors d'un enregistrement

Si nous indiquons NOT NULL, le SGBD attendra de recevoir une valeur lors de la demande d'insertion d'un nouvel enregistrement, faute de quoi il nous retournera une erreur.

Au contraire, DEFAULT NULL mettra alors 'null' comme valeur par défaut si le champs n'est pas complété.

```
1 CREATE DATABASE test1234;
2
3 USE test1234;
4
5 CREATE TABLE `users` (
6     id_user int not null,
7     name_user varchar(50) not null,
8     password_user varchar(100) not null,
9     mail_user varchar(150) not null,
10    age_user date not null,
11    id_town int default null
12 );
```




Exercice d'application n°1 : Création BDD + Table

- Créez la base de donnée test1234 sur votre serveur
- Ajoutez la table users au sein de votre BDD, telle que définie sur les diapos précédentes
- Ajoutez une nouvelle table `towns` ayant pour champs :
 - id_town : entier
 - name_town : varchar
 - cp_town : varchar

Vous devrez choisir une taille adéquate, et un comportement logique pour le nullable



Solution Exercice N°1

```
CREATE TABLE `towns`(  
  id_town int not null,  
  name_town varchar(100) not null,  
  cp_town varchar(20) not null  
);
```



Requête ALTER TABLE

La requête ALTER TABLE permet de modifier les tables de votre BDD

```
ALTER TABLE users
```




Requête ALTER TABLE

Cette requête permet de modifier une table (alter table, altérer/modifier une table) pour y ajouter (add/ajouter) un argument (primary key).

Ici, nous ajoutons une PRIMARY KEY, clé primaire (PK).

Cette dernière est et doit être unique et est attribuée au champ 'id_user'.

```
ALTER TABLE users  
ADD PRIMARY KEY (id_user);
```

Effectuez la requête sur la table `town` également



Requête ALTER TABLE

Nous ajoutons une contrainte sur notre table 'users' avec le terme ADD CONSTRAINT.

Cette contrainte sera une FOREIGN KEY (FK) ou « clé étrangère ».

Cette FK correspondra au champ 'id_town' de la table 'users' et comme nous l'indique REFERENCES, elle fera donc références à la table 'towns' et précisément à son champ 'id_town'.

```
ALTER TABLE users  
ADD CONSTRAINT FOREIGN KEY (id_town) REFERENCES towns (id_town);
```

Pour appliquer une contrainte de clé étrangère, il faut que le champ de référence soit définie comme clé primaire

!



Requête ALTER TABLE

Sur PHPMyAdmin, cliquez sur le nom de votre BDD dans la liste de gauche.

Cliquez ensuite sur l'onglet "Plus" en haut à droite, puis sur l'onglet "Concepteur"

v	test1234 users
🔑	id_user : int(11)
📋	name_user : varchar(50)
📋	password_user : varchar(100)
📋	mail_user : varchar(150)
📅	age_user : date
#	id_town : int(11)

v	test1234 towns
🔑	id_town : int(11)
📋	name_town : varchar(100)
📋	cp_town : varchar(20)



Requête ALTER TABLE

L'argument 'MODIFY' permet de modifier un champ de notre table.

Il faudra préciser le nom du champ, son type, ainsi que ses différents paramètres.

Ici, 'id_user' sera donc un « int », « non nul », qui va s'auto-incrémenter. (Faire « +1 » à chaque nouvel enregistrement sur la valeur de la clé)

```
ALTER TABLE users  
MODIFY id_user int not null AUTO_INCREMENT;
```

Modifier un champ ne modifie pas les contraintes et clés mises en place



Requête ALTER TABLE

L'argument 'ADD COLUMN' permet d'ajouter une column

```
ALTER TABLE users  
ADD COLUMN phone varchar(30);
```

L'argument 'CHANGE' permet de changer une column

```
ALTER TABLE users  
CHANGE phone phone_user varchar(20);
```

ATTENTION : Diminuer la taille d'un champ entrainera la découpe des données



Requête ALTER TABLE

L'argument 'DROP COLUMN' (ou DROP) permet de supprimer une column

```
ALTER TABLE users  
DROP phone;
```

```
ALTER TABLE users  
DROP COLUMN phone;
```

ATTENTION : Toutes les données présentes dans la column seront irrémédiablement perdues



Requête DROP TABLE

La requête 'DROP TABLE' permet de supprimer une table de la BDD

```
DROP TABLE MACHIN;
```

ATTENTION : Toutes les données présentes dans la table seront irrémédiablement perdues



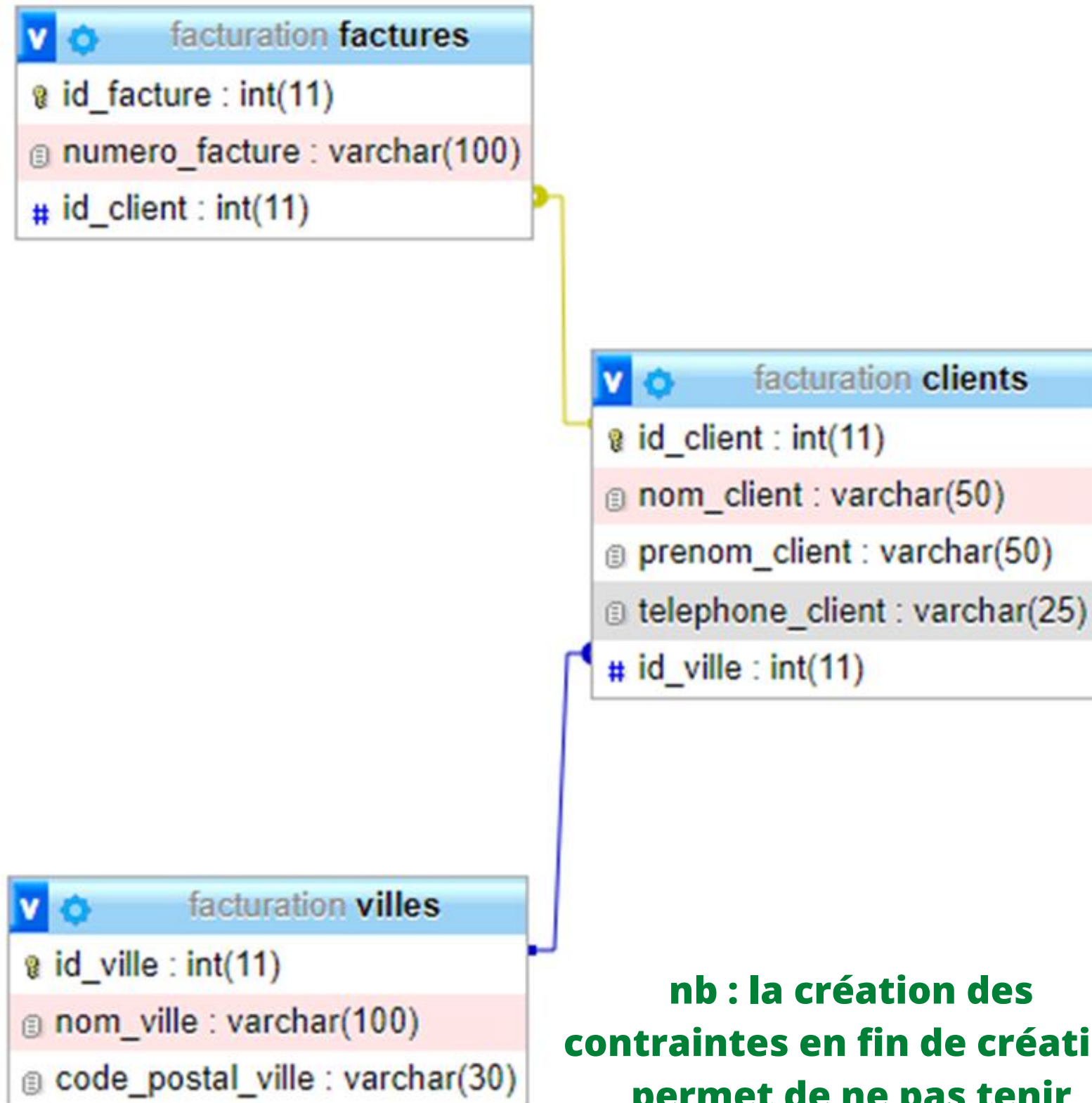
Exercice d'application n°2 : Requêtes de Structure

- Créez une BDD 'facturation' contenant 3 tables : Clients, Facturations, Villes
- La table client aura 5 champs : id_client, nom_client, prenom_client, telephone_client, id_ville
 - La table factures aura 3 champs : id_facture, numero_client, id_client
 - La table villes aura 3 champs : id_ville, nom_ville, cp_ville

Vous devrez choisir une taille adéquate, un type adéquate, un comportement logique pour le nullable, et mettre en place les relations clés primaires / clés secondaires

Exercice d'application n°2 : Correction

32



nb : la création des contraintes en fin de création permet de ne pas tenir compte de l'ordre de création des tables

```
CREATE DATABASE facturation;

USE facturation;

CREATE TABLE villes(
    id_ville int NOT null PRIMARY KEY AUTO_INCREMENT,
    nom_ville varchar(100) not null,
    cp_ville varchar(30)
);

CREATE TABLE clients(
    id_client int NOT null PRIMARY KEY AUTO_INCREMENT,
    nom_client varchar(50) not null,
    telephone_client varchar(30),
    id_ville int not null
);

CREATE TABLE factures(
    id_facture int NOT null PRIMARY KEY AUTO_INCREMENT,
    numero_facture varchar(100) not null,
    id_client int not null
);

ALTER TABLE factures
ADD CONSTRAINT FOREIGN KEY (id_client) REFERENCES clients (id_client);

ALTER TABLE clients
ADD CONSTRAINT FOREIGN KEY (id_ville) REFERENCES villes (id_ville);
```



Relations multiples

Nous avons étudié des requêtes sur des tables avec lesquelles nous avons que des relations (0,1) / (0,n).

Pour expliquer la relation entre la table clients et la table villes, nous pourrions dire qu'un client peut habiter dans aucune ville si celle-ci n'est pas enregistrée dans la BDD et jusqu'à 1 seule ville au maximum. (0,1)

A l'inverse, une ville peut-être habitée par aucun client et jusqu'à 'n' clients, s'il y a plusieurs clients qui vivent dans la même ville. (0,n)

Mais que faire dans une relation (0,n) / (0,n) ?



Relations multiples

Pour une relation en (0,n) / (0,n), il faut mettre en place une table d'association.

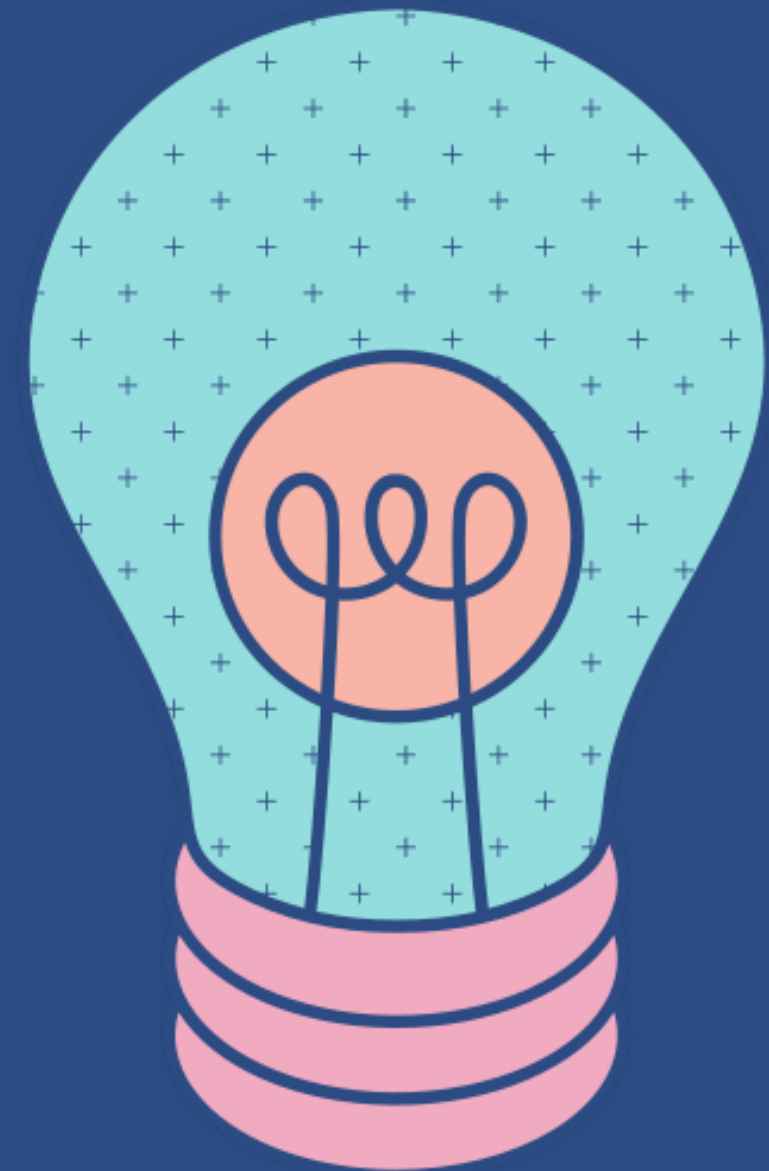
Avant cela, il faut bien sûr créer les tables avec lesquelles elle sera associée.

Ensuite, notre table d'association accueillera les clés étrangères des 2 autres tables, qui associées, feront sa clé primaire.

```
CREATE TABLE villes(  
    id_ville int not null PRIMARY KEY AUTO_INCREMENT,  
    nom_ville varchar(50) not null  
);  
  
CREATE TABLE codes(  
    id_cp int not null PRIMARY KEY AUTO_INCREMENT,  
    cp_code varchar(30) not null  
);  
  
CREATE TABLE correspondre(  
    id_ville int,  
    id_cp int,  
    PRIMARY KEY (id_ville, id_cp),  
    FOREIGN KEY (id_ville) REFERENCES villes (id_ville),  
    FOREIGN KEY (id_cp) REFERENCES codes (id_cp)  
);
```

Requêtes de Données

```
view plain copy to clipboard print ?
01. SELECT
02.     DB.datname                                as DatabaseId
03.     ,md5(alldb.normalize_query(ST.query))      as QueryId
04.     ,alldb.normalize_query(ST.query)          as NQuery
05.     ,sum(total_time)                          as totaltime
06.     ,sum(rows)                                as totalrows
07.     ,sum(calls)                              as totalcalls
08.     ,sum(ST.shared_blks_hit)                  as totalshhits
09.     ,sum(ST.shared_blks_read)                 as totalshread
10.     ,sum(ST.shared_blks_written)              as totalshwritten
11. FROM pg_stat_statements ST
12. JOIN pg_database DB on DB.oid = ST.dbid
13. GROUP BY 1,2,3
14. having sum(ST.shared_blks_hit) > 1500
15. order by 4 desc LIMIT 10 ;
```





requête **INSERT TO**

Cette méthode nous permettra d'insérer des données dans une base en créant un nouvel enregistrement (ou tuple) .

Elle a 2 syntaxes principales.



requête INSERT TO

1) Insérer une ligne en indiquant les informations pour toutes les colonnes existantes (en respectant l'ordre)

INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...)

Avec celle-ci nous avons les avantages et inconvénients suivants:

- Obliger de remplir toutes les données
 - Respecter l'ordre des colonnes
 - Moins de fautes de frappes
- Colonnes peuvent être renommées

```
INSERT INTO users VALUES (0,'john','fsqiufhsqivdhovdvfds16532','john@gmail.com',17,1)
```



requête **INSERT TO**

2) Insérer une ligne en spécifiant les colonnes que vous souhaitez compléter.
Il est possible d'insérer seulement une partie des champs

INSERT INTO table (nom_colonne_1, nom_colonne_2, ...) **VALUES** ('valeur 1',
'valeur 2', ...)

Avec elle, il est possible:

- De ne pas renseigner toutes les colonnes.
- L'ordre des colonnes n'est pas important.

```
INSERT INTO users (id_user,name_user,password_user,mail_user,age_user) VALUES  
(2,'tom','gsfduivdk15455dsfgsdfbbvs6d54sdf','tom@gmail.com',32)
```



requête INSERT TO

Il est également possible d'insérer plusieurs enregistrements en une seule requête, peut importe la méthode, voici un exemple :

```
INSERT INTO users (password_user, name_user, id_ville) VALUES  
("monMotDePasse", "monNom", 31000),  
("monMotDePasse2", "monNom2", null),  
("monMotDePasse3", "monNom3", 75000),  
("monMotDePasse4", "monNom4", null);
```




requête UPDATE

Cette méthode nous autorisera à effectuer des modifications dans une base sur des enregistrements déjà existants.

Cette commande est utilisé avec « WHERE », qui est une condition permettant d'extraire des enregistrements d'une table.

```
UPDATE users SET name_user = 'thomas' WHERE id_user=2
```

NB : Si aucune condition n'est ajoutée, toutes les données sont alors modifiées

requête UPDATE

```
UPDATE users  
SET idTown = "11"  
WHERE idUser = 3;
```

idUser	idsession	name	password	mail	phone	idtown
1	NULL	test1	pass1	NULL	NULL	33330
2	NULL	test2	pass2	NULL	NULL	NULL
3	NULL	test3	pass3	NULL	NULL	11
4	NULL	test4	pass4	NULL	NULL	NULL

```
UPDATE users  
SET idTown = "74";
```

idUser	idsession	name	password	mail	phone	idtown
1	NULL	test1	pass1	NULL	NULL	74
2	NULL	test2	pass2	NULL	NULL	74
3	NULL	test3	pass3	NULL	NULL	74
4	NULL	test4	pass4	NULL	NULL	74



requête DELETE

Cette commande permet de supprimer des enregistrements dans une table.

En associant cette commande avec « WHERE », il est donc possible de sélectionner seulement les lignes soumises à la condition.

**ATTENTION : Toutes les données sélectionnées seront irrémédiablement perdues !!
Il est fortement conseillé d'effectuer une sauvegarde de la BDD avant suppression**

!

requête DELETE

```
1 DELETE FROM users WHERE id_user=1
```

←T→		id_user	name_user	password_user	mail_user	age_user	id_town
<input type="checkbox"/>	✎ Éditer	2	thomas	gsfduivdk15455dsfgsdfbbvs6d54sdf	tom@gmail.com	0000-00-00	NULL

ATTENTION : En absence de WHERE , toutes les données seront supprimées

!



DELETE FROM

VS

TRUNCATE TABLE

- DELETE FROM `table` permet de supprimer l'intégralité des données si aucune condition n'est spécifiée mais préserve l'index
- TRUNCATE TABLE `table` permet de supprimer l'intégralité des données et de remettre à zéro l'index



Exercice d'application n°3 : partie 1/3

Requêtes de Structure & données

- Vous êtes responsable d'une société de location de voitures, et avez sous votre responsabilité :
 - des responsables d'agence, des locaux situés dans diverses villes et une flotte de voitures
- Les responsables d'agences ont pour attributs : nom, prenom, tel, mail
 - Locaux : adresse (rue, ville, cp)
- Voitures : Etat location (loué ou non), marque, couleur. Elles peuvent être louées et rendues dans n'importe quelle agence

Vous devez rajoutez au moins 4 données par table, et faire MCD/MLD



Exercice d'application n°3 : partie 2/3

Requêtes de Structure & données

- Vous avez fait l'acquisition d'une nouvelle agence située avenue bloblocar, 22 222, tikilou.
Reponsable : FourWheels, tel : 88.11.55.22.33, mail : toukilou@atoutroule.fr
 - Acheté 2 nouvelles voitures : fiat 7014 jaune; farp kagi orange
- La 3e voiture crée sur le première partie a eu un accident, et change en couleur marron clair
 - le nom du champ marque ne convient pas, et doit être changé en 'marque_et_modele'
 - Les nouvelles adresses emails impliquent une augmentation de la taille du champ mail
 - Le champ 'louer_ou_pas' n'est finalement pas nécessaire, supprimez le
 - La seconde voiture créé ne roulera plus cause accident, retirez la

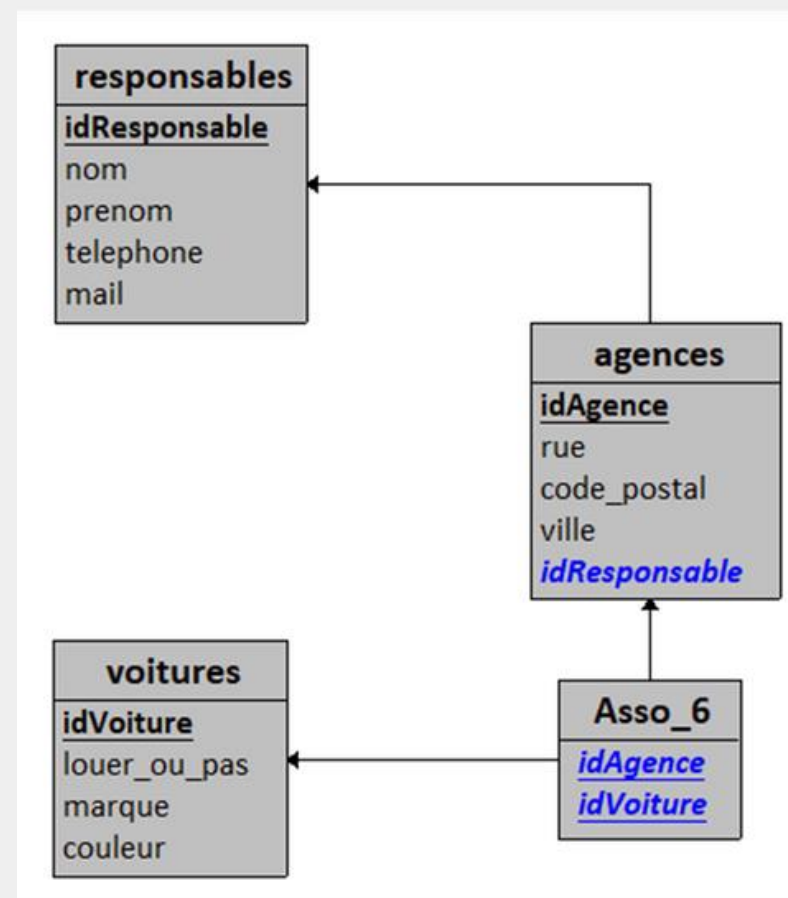


Exercice d'application n°3 : partie 3/3 (facultatif)

Requêtes de Structure & données

- Le responsable de FourWheels Joulou a changé de téléphone, et sera maintenant joignable au : 46.25.63.14.98. Modifiez le avec une condition sur nom et prénom.
- Ajoutez une colonne modèle dans la table voitures, puis renommez 'marque_et_modele' en marque
- Modifiez la 1e voiture en changeant sa couleur en marron clair, puis supprimez toutes les voitures marrons clair

Exercice d'application n°3 : Correction



```

create database exo;
use exo;
CREATE TABLE responsables(
    idResponsable INT PRIMARY KEY auto_increment,
    nom VARCHAR(50),
    prenom VARCHAR(50),
    telephone VARCHAR(50),
    mail VARCHAR(100)
);
CREATE TABLE agences(
    idAgence INT PRIMARY KEY auto_increment,
    rue VARCHAR(200),
    code_postal VARCHAR(100),
    ville VARCHAR(100),
    idResponsable INT,
    FOREIGN KEY(idResponsable) REFERENCES responsables(idResponsable)
);
CREATE TABLE voitures(
    idVoiture INT PRIMARY KEY auto_increment,
    louer_ou_pas smallint,
    marque VARCHAR(50),
    couleur VARCHAR(50)
);
CREATE TABLE abriter(
    idAgence INT,
    idVoiture INT,
    PRIMARY KEY(idAgence, idVoiture),
    FOREIGN KEY(idAgence) REFERENCES agences(idAgence),
    FOREIGN KEY(idVoiture) REFERENCES voitures(idVoiture)
);
  
```


Exercice d'application n°3 : Correction

```
insert into voitures (louer_ou_pas, marque, couleur) values
("0", "voit1", "rouge"),
("0", "voit2", "blanche"),
("1", "voit3", "bleu"),
("0", "voit4", "noire");
insert into responsables (nom, prenom, telephone, mail) values
("Radial", "Paul", "77.88.99.88.14", "jrroule@atoutroule.fr"),
("Royaluni", "Henri", "25.63.41.69.49", "tataouine@atoutroule.fr"),
("Michemiche", "Pierre", "34.64.21.64.78", "losbagnos@atoutroule.fr"),
("Conti", "Arthur", "98.76.15.34.15", "etailleurs@atoutroule.fr");
insert into agences (rue, code_postal, ville) values
("rue Atouteblinde", "36521", "Trop-de-la-balle"),
("avenue Vive-allure", "62541", "Marreteplus"),
("impasse limitation-de-vitesse", "44551", "Mincealors"),
("boulevard Grosexcès", "77889", "Jen-peu-plus");
```

```
insert into agences (rue, code_postal, ville) values
("avenue bloblocar", "22222", "Toukilou");
```

```
insert into responsables (nom, prenom, telephone, mail) values
("FourWheels", "Joulou", "64.15.42.19.76", "toukilou@atoutroule.fr");
```

```
insert into voitures (marque, couleur) values
("fiot 7014", "jaune"),
("farp kagi", "orange");
```


Exercice d'application n°3 : Correction

```
update voitures  
set couleur = "marron clair"  
where idVoiture = 3;
```

```
alter table voitures  
change marque marque_et_modele varchar(100);
```

```
alter table responsables  
modify mail varchar(255);
```

```
alter table voitures  
drop louer_ou_pas;
```

```
delete from voitures  
where idVoiture = 2;
```

```
update responsables  
set telephone = "46.25.63.14.98"  
where nom = "FourWheels"  
and prenom = "Joulou";
```

```
alter table voitures  
add column modele varchar(50);
```

```
alter table voitures  
change marque_et_modele marque varchar(50);
```

```
update voitures  
set couleur = "marron clair"  
where idVoiture = 1;
```

```
delete from voitures  
where couleur = "marron clair";
```



Requête SELECT

La commande SELECT permet de faire des opérations de recherche à partir des enregistrements d'un ou plusieurs attributs d'une table:

SELECT nom_attribut FROM table1 ;

Et pour sélectionner plusieurs attributs on les sépare par des virgules:

SELECT nom_attribut1, nom_attribut2 FROM table1 ;

Et pour sélectionner tous les attributs :

*SELECT * FROM table1 ;*



Requête **SELECT DISTINCT**

La commande **SELECT** permet d'afficher des enregistrements en double s'ils existent.

La commande **DISTINCT** permet d'éviter des redondances dans les résultats.
SELECT DISTINCT nom_attribut FROM table1 ;



Requête **SELECT AS**

La commande « AS » permet d'utiliser des alias pour renommer temporairement un attribut ou une table dans une requête.

SELECT nom_attribut1 AS na1, nom_attribut2 AS na2 FROM table1 ;

==

SELECT nom_attribut1 na1, nom_attribut2 na2 FROM table1 ;

Pour une table

SELECT nom_attribut1 FROM table1 AS t1 ;

Opérateurs de comparaison

Opérateur	Description
=	Egale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
IN	Liste de plusieurs valeurs possibles
BETWEEN AND	Valeur comprise dans un intervalle de données
LIKE	Recherche en spécifiant le début, le milieu ou la fin d'un mot
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle



Opérateurs Logiques AND/OR

Les opérateurs logiques AND et OR peuvent être utilisés dans la commande WHERE pour combiner des conditions entre elles.

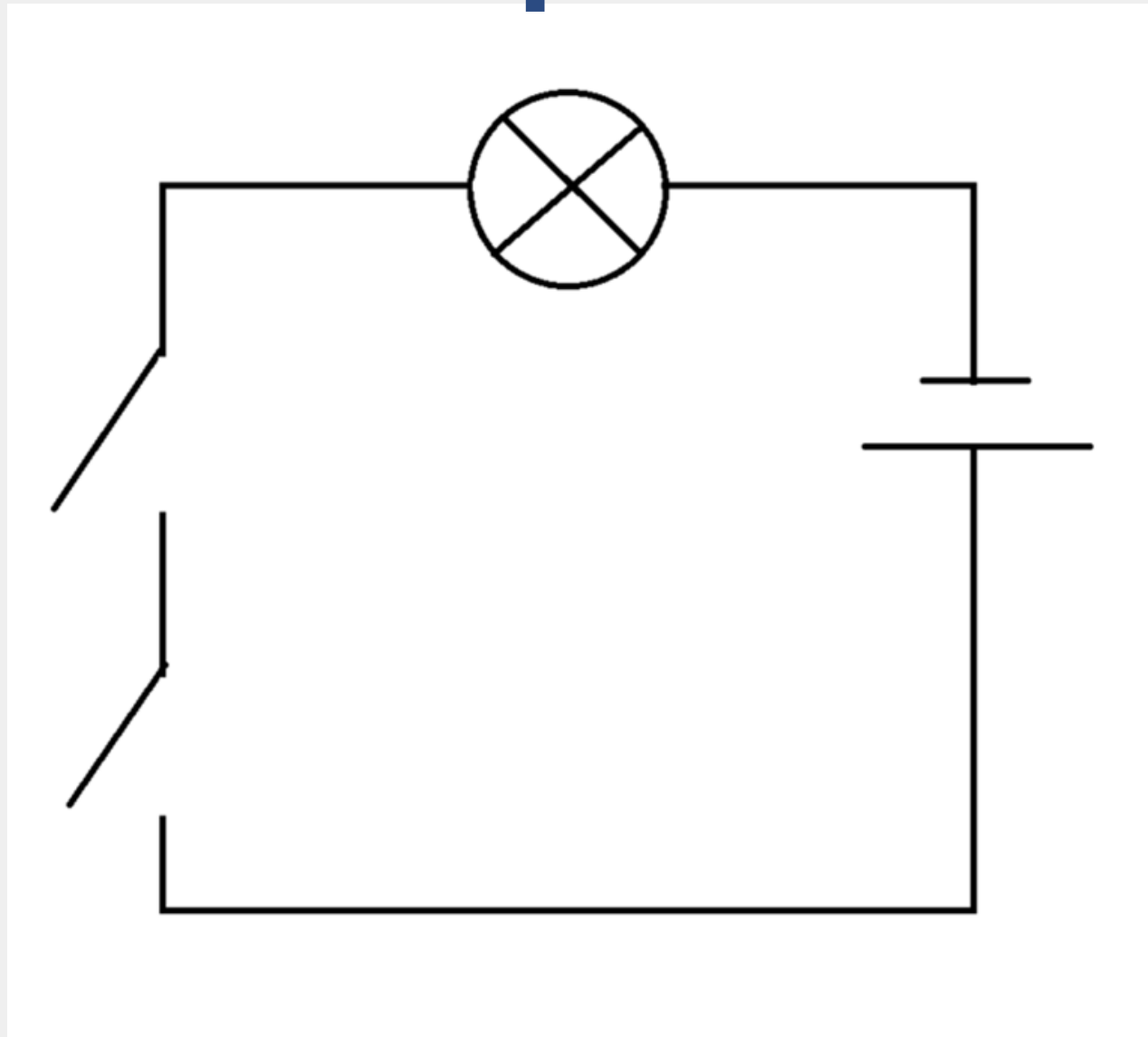
L'opérateur And dont la syntaxe est :

SELECT nom_attribut1, nom_attribut2 FROM table1 WHERE condition1 AND condition2;

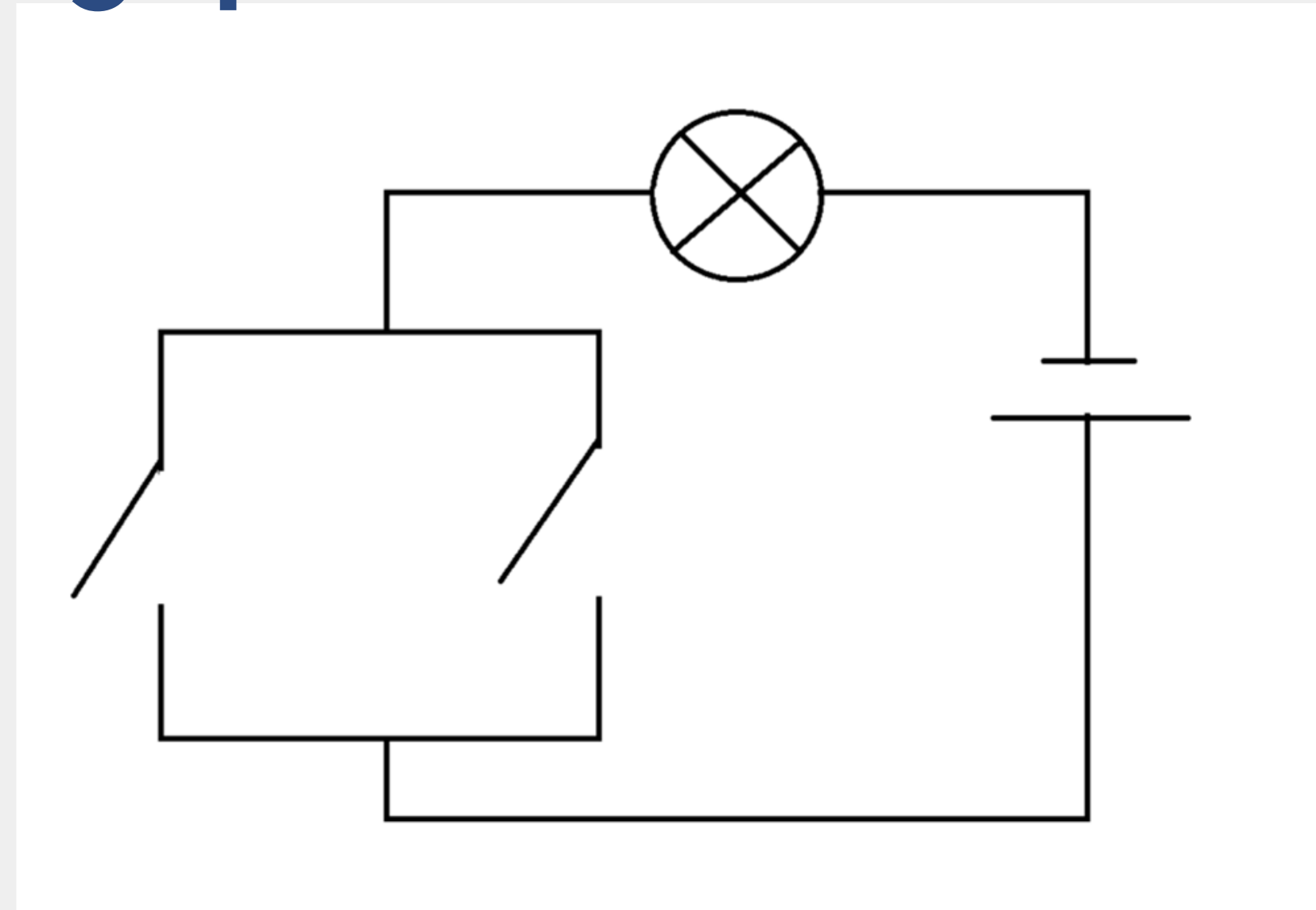
L'opérateur Or dont la syntaxe est :

SELECT nom_attribut1, nom_attribut2 FROM table1 WHERE condition1 OR condition2;

Opérateurs Logiques AND/OR



AND



OR



Opérateurs Logiques AND/OR

Il est également possible de combiner ces 2 opérateurs logiques ce qui pourrait donner une syntaxe comme celle-ci par exemple:

```
SELECT nom_attribut1, nom_attribut2  
FROM table1  
WHERE condition1 AND (condition2 OR condition3);
```




Commande ORDER BY

La commande ORDER BY permet de trier des lignes dans un résultat d'une requête. Il est possible de trier les données sur un ou plusieurs attributs par ordre ascendant ou descendant.

```
SELECT nom_attribut1, nom_attribut2  
FROM table1  
ORDER BY nom_attribut1
```

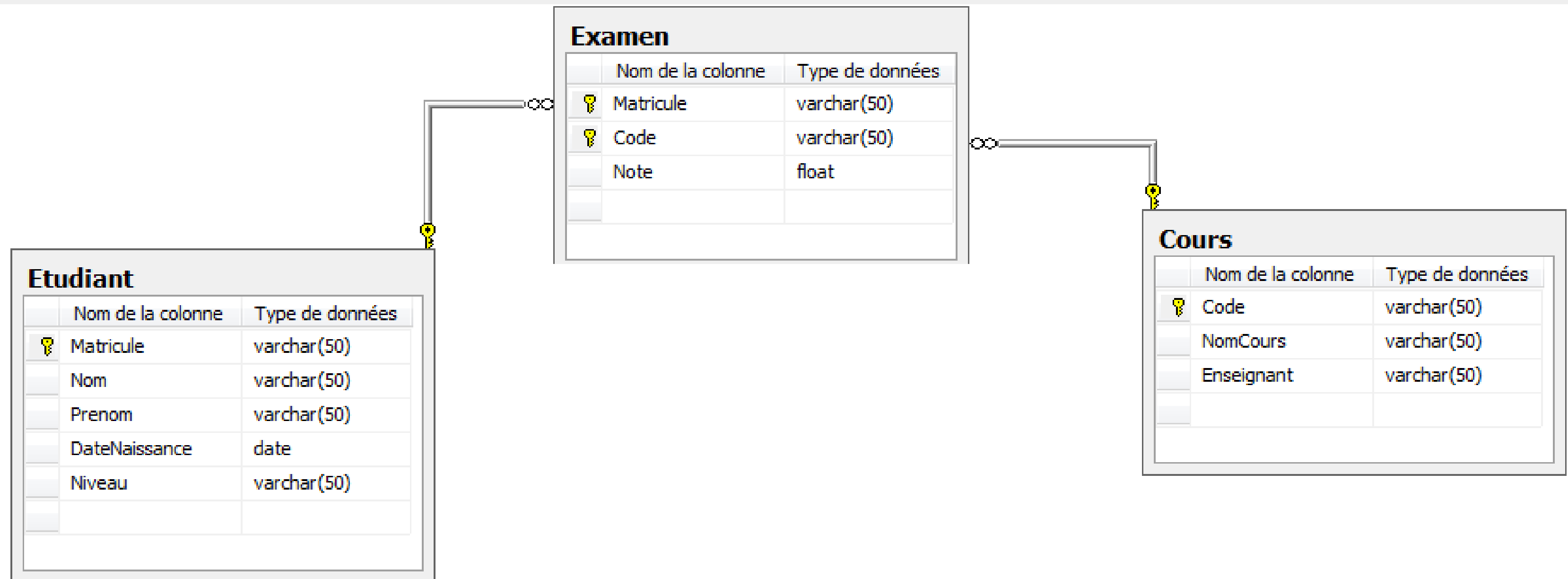
Par défaut, les résultats sont classés par ordre ascendant.

Pour trier le résultat par ordre décroissant, il faut ajouter DESC après le champ à trier (ORDER BY nom_attr1 DESC)

Exercice d'application n°4 : 1/2

Sélection et ordonnancement de données

- Créez la BDD Etudiants, contenant les tables et relations suivantes :





Exercice d'application n°4 : 2/2

Sélection et ordonnancement de données

- Afficher la liste des étudiants triés par ordre croissant de date de naissance.
- Afficher tous les étudiants inscrits à M1 et tous les étudiants inscrits à M2.
- Afficher les matricules des étudiants qui ont passé l'examen du cours 002.
- Afficher les matricules de tous les étudiants qui ont passé l'examen du cours 001 et de tous les étudiants qui ont passé l'examen du cours 002.
- Afficher le matricule, code, note /20 et note /40 de tous les examens classés par ordre croissant de matricule et de code.
 - Trouver la moyenne de notes de cours 002.
- Compter les examens passés par un étudiant (exemple avec matricule 'e1')
 - Compter le nombre d'étudiants qui ont passé l'examen du cours 002.
- Calculer la moyenne des notes d'un étudiant (exemple avec matricule 'e1').
 - Compter les examens passés par chaque étudiant.
 - Calculer la moyenne des notes pour chaque étudiant.
 - Trouver la moyenne de notes de chaque cours.



Exercice d'application n°4 : Solution

Créer la BDD Etudiants

```
CREATE DATABASE etudiants;

USE etudiants;

CREATE TABLE Etudiant(
    Matricule varchar(50) NOT null PRIMARY KEY,
    Nom varchar(50),
    Prenom varchar(50),
    DateNaissance date,
    Niveau varchar(50)
);

CREATE TABLE Cours(
    Code varchar(50) NOT null PRIMARY KEY,
    NomCours varchar(50),
    Enseignant varchar(50)
);

CREATE TABLE Examen(
    Matricule varchar(50) NOT null,
    Code varchar(50) NOT null,
    Note float,
    PRIMARY KEY (Matricule,Code),
    FOREIGN KEY (Matricule) REFERENCES Etudiant(matricule),
    FOREIGN KEY (Code) REFERENCES Cours(Code)
);
```



Exercice d'application n°4 : Solution

```
--Q1  
select *  
from Etudiant  
order by DateNaissance
```

```
--Q2  
select *  
from Etudiant  
where Niveau='M1' or Niveau='M2'
```

```
--Q3  
select Matricule  
from Examen  
where Code='002'
```

```
--Q4  
select Matricule  
from Examen  
where Code='001' or Code='002'
```

```
--Q5  
select Matricule,Code,Note as "Note sur 20",Note*2 as "Note sur 40"  
from Examen  
order by Matricule,Code
```

```
--Q6  
select AVG(Note) as "Moyenne de notes"  
from Examen  
where Code='002'
```

```
--Q7  
select COUNT(Code) as "Nombre d'examens"  
from Examen  
where Matricule='e1'
```

```
--Q8  
select COUNT(Matricule) as "Nombre d'étudiants"  
from Examen  
where Code='002'
```

```
--Q9  
select AVG(Note) as "Moyenne de notes"  
from Examen  
where Matricule='e1'
```

```
--Q10  
select Matricule,COUNT(Code) as "Nombre d'examens"  
from Examen  
group by Matricule
```

```
--Q11  
select Matricule,AVG(Note) as "Moyenne de notes"  
from Examen  
group by Matricule
```

```
--Q12  
select Code,AVG(Note) as "Moyenne de notes"  
from Examen  
group by Code
```



Jointures de tables

Les jointures permettent d'associer plusieurs tables dans une requête en utilisant la clé étrangère.

Il existe plusieurs méthodes pour associer 2 tables ensemble:

INNER JOIN

CROSS JOIN

LEFT JOIN (ou LEFT OUTER JOIN)

RIGHT JOIN (ou RIGHT OUTER JOIN)

FULL JOIN (ou FULL OUTER JOIN)

SELF JOIN

NATURAL JOIN

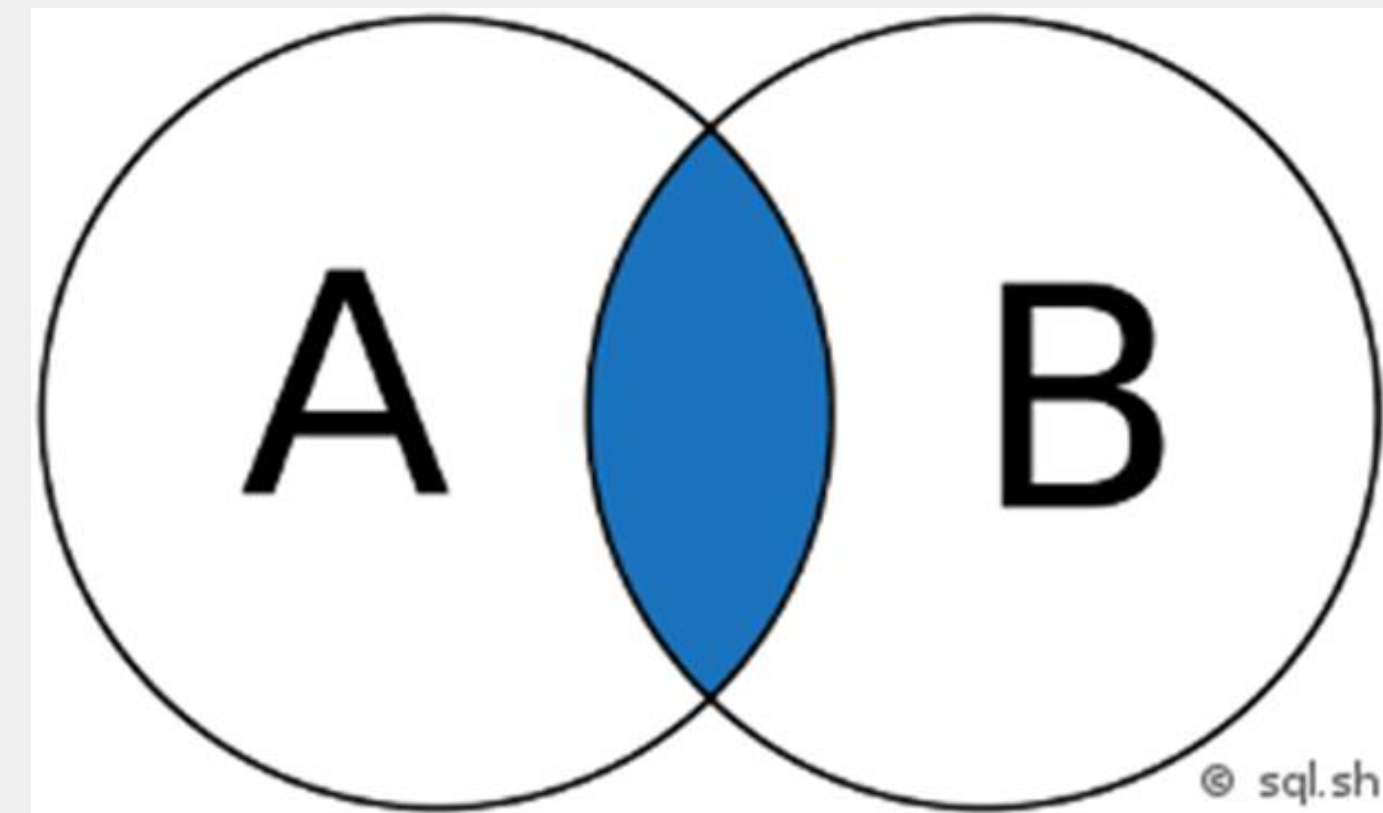


INNER JOIN

INNER JOIN : jointure pour retourner les enregistrements quand la condition est vrai dans les 2 tables.

C'est la jointure la plus commune.

```
SELECT *  
FROM A  
INNER JOIN B ON A.key =  
B.key
```





INNER JOIN

id_util	nom_util	id_droit
1	Toto	1
2	Titi	2
3	Tata	3
4	John	null
5	Joe	null

id_droit	nom_droit
1	Admin
2	Modo
3	Editeur
4	Utilisateur
5	Visiteur

id_util	nom_util	nom_droit
1	Toto	Admin
2	Titi	Modo
3	Tata	Editeur

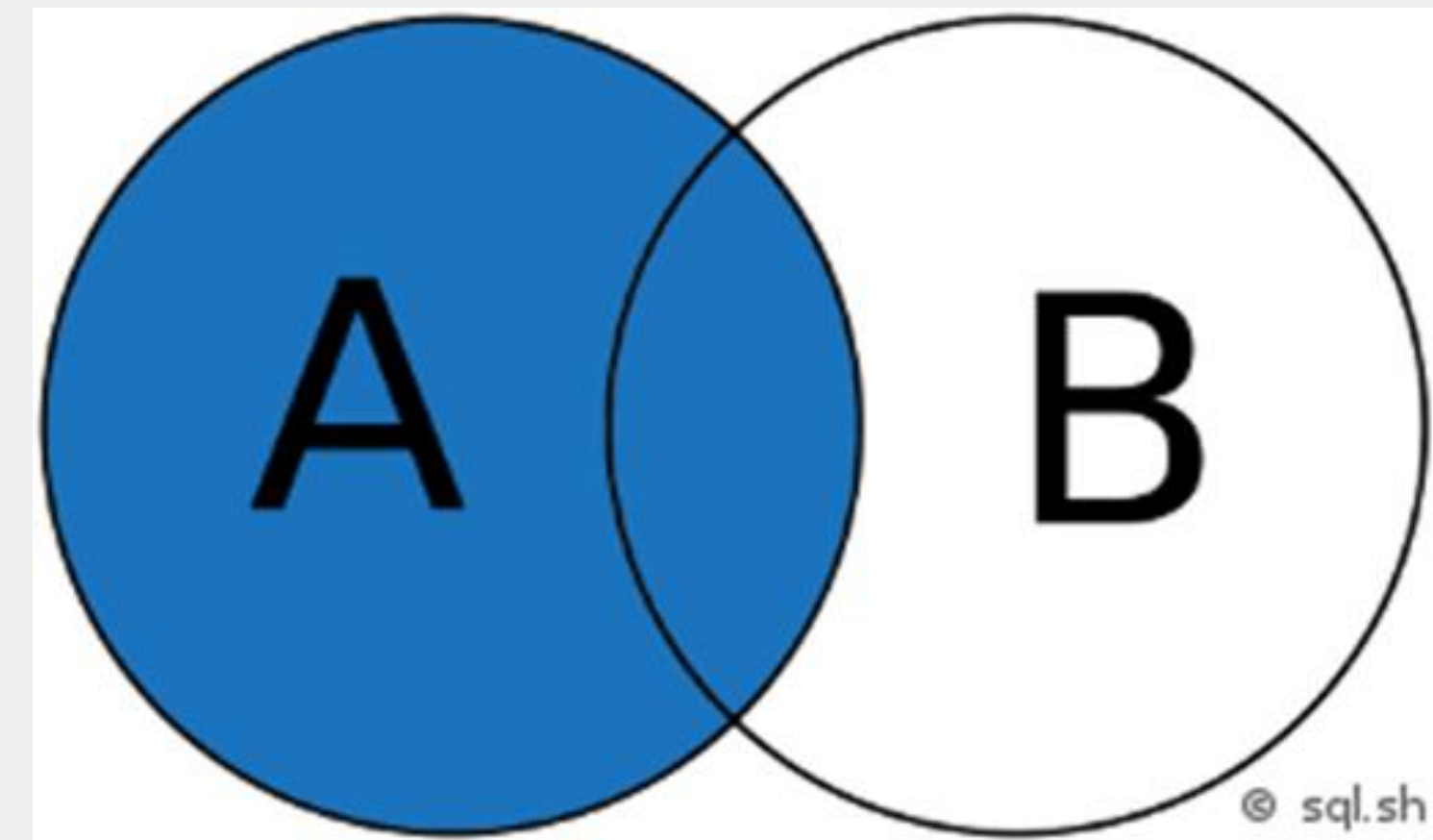
SELECT * FROM util
INNER JOIN droit ON util.id_droit= droit.id_droit



LEFT JOIN

LEFT JOIN : jointure pour retourner tous les enregistrements de la table de gauche (Première table de la requête) même si la condition n'est pas vérifiée dans l'autre table.

```
SELECT *  
FROM A  
LEFT JOIN B ON A.key =  
B.key
```



LEFT JOIN

id_util	nom_util	id_droit
1	Toto	1
2	Titi	2
3	Tata	3
4	John	null
5	Joe	null

id_droit	nom_droit
1	Admin
2	Modo
3	Editeur
4	Utilisateur
5	Visiteur

id_util	nom_util	nom_droit
1	Toto	Admin
2	Titi	Modo
3	Tata	Editeur
4	John	Null
5	Joe	Null

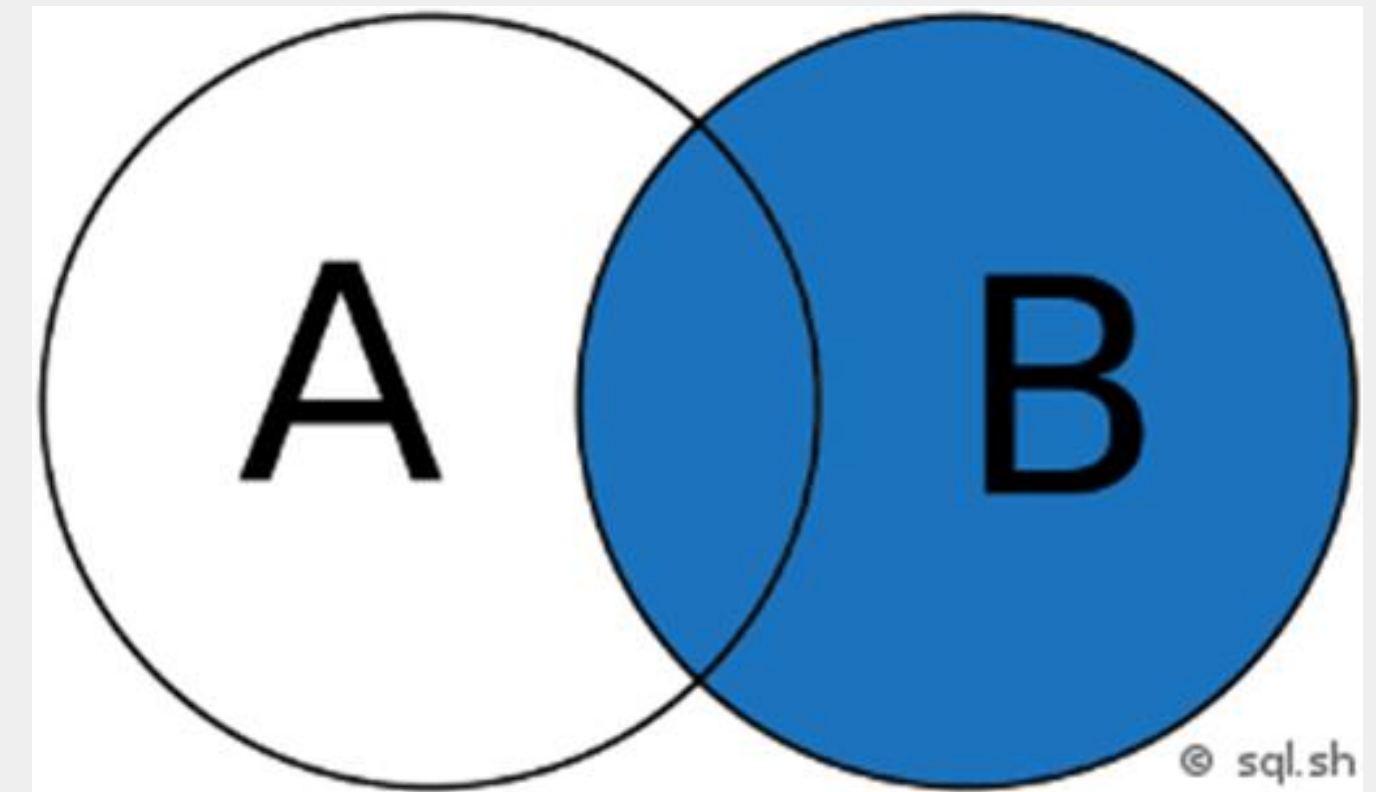
SELECT * FROM util
LEFT JOIN droit ON util.id_droit= droit.id_droit



RIGHT JOIN

RIGHT JOIN : jointure pour retourner tous les enregistrements de la table de droite (Deuxième table de la requête) même si la condition n'est pas vérifié dans l'autre table.

```
SELECT *  
FROM A  
RIGHT JOIN B ON A.key =  
B.key
```



RIGHT JOIN

id_util	nom_util	id_droit
1	Toto	1
2	Titi	2
3	Tata	3
4	John	null
5	Joe	null

id_droit	nom_droit
1	Admin
2	Modo
3	Editeur
4	Utilisateur
5	Visiteur

id_util	nom_util	nom_droit
1	Toto	Admin
2	Titi	Modo
3	Tata	Editeur
null	null	Utilisateur
null	null	Visiteur

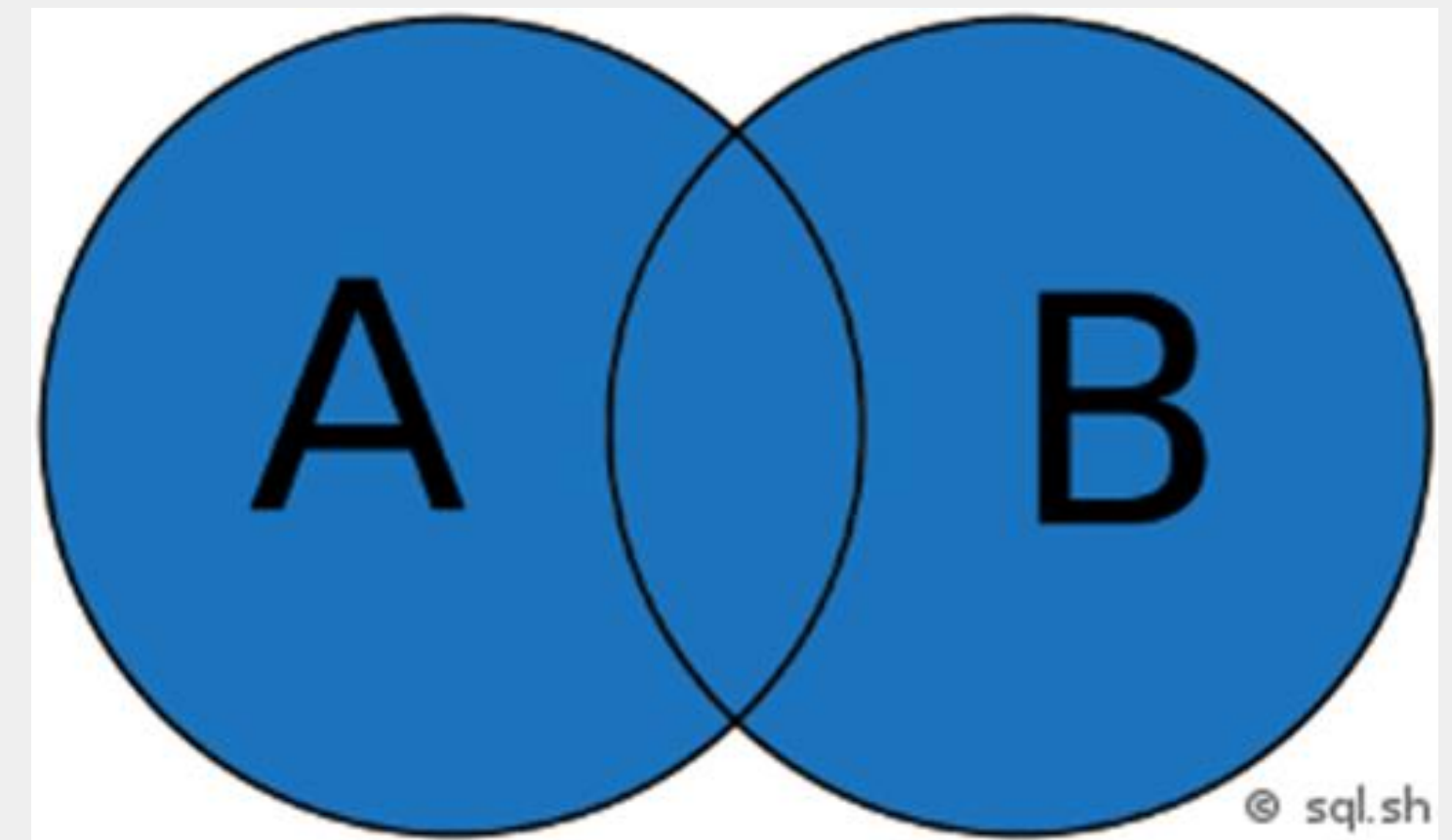
SELECT * FROM util
RIGHT JOIN droit ON util.id_droit= droit.id_droit



FULL JOIN

FULL JOIN : jointure pour retourner les résultats quand la condition est vrai dans au moins une des 2 tables.

```
SELECT *  
FROM A  
FULL JOIN B ON A.key =  
B.key
```



© sql.sh

RIGHT JOIN

id_util	nom_util	id_droit
1	Toto	1
2	Titi	2
3	Tata	3
4	John	null
5	Joe	null

id_droit	nom_droit
1	Admin
2	Modo
3	Editeur
4	Utilisateur
5	Visiteur

id_util	nom_util	nom_droit
1	Toto	Admin
2	Titi	Modo
3	Tata	Editeur
4	John	Null
5	Joe	Null
null	null	Utilisateur
null	null	Visiteur

SELECT * FROM util

FULL JOIN droit ON util.id_droit= droit.id_droit



Requêtes imbriquées

Les Requêtes imbriquées:

Cela consiste à exécuter une requête à l'intérieur d'une autre requête.

EXEMPLE:

```
INSERT INTO `utilisateur` (nom_util, id_droit) VALUES  
("toto", (SELECT id_droit FROM `droits` WHERE nom_droit = "admin"))
```




GROUP BY

La commande « GROUP BY » est utilisée avec les fonctions de calcul.

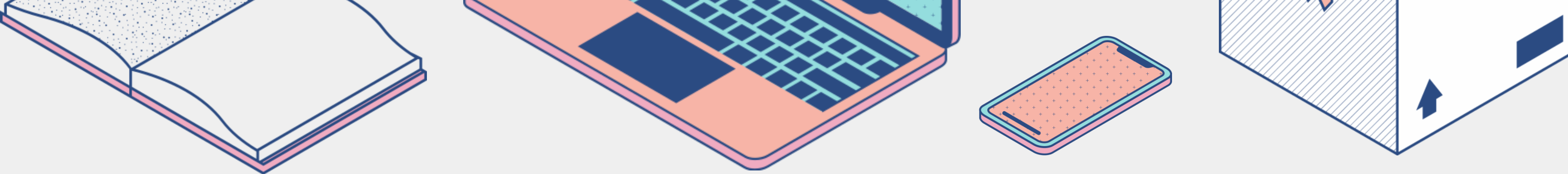
Elle permet de regrouper le résultat.

Elle est obligatoire sur tous les attributs qui entourent la fonction de calcul dans le SELECT.

```
SELECT nom_attribut1, fonction (nom_attribut2)
```

```
FROM table1
```

```
GROUP BY nom_attribut1;
```



GROUP BY

Cette commande doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

Les fonctions possibles :

Fonctions	Description
AVG()	Calcul de la moyenne d'un ensemble de valeurs
COUNT()	Pour compter le nombre de lignes concernées
MAX()	Permet de récupérer la plus haute valeur
MIN()	Permet de récupérer la plus petite valeur
SUM()	Somme de plusieurs lignes



HAVING

Cette commande est presque similaire au WHERE à la seule différence que le HAVING permet de filtrer en utilisant les fonctions de calcul.

```
SELECT nom_attribut1, fonction (nom_attribut2)
FROM table1
GROUP BY nom_attribut1
HAVING fonction (nom_attribut2) = condition1;
```



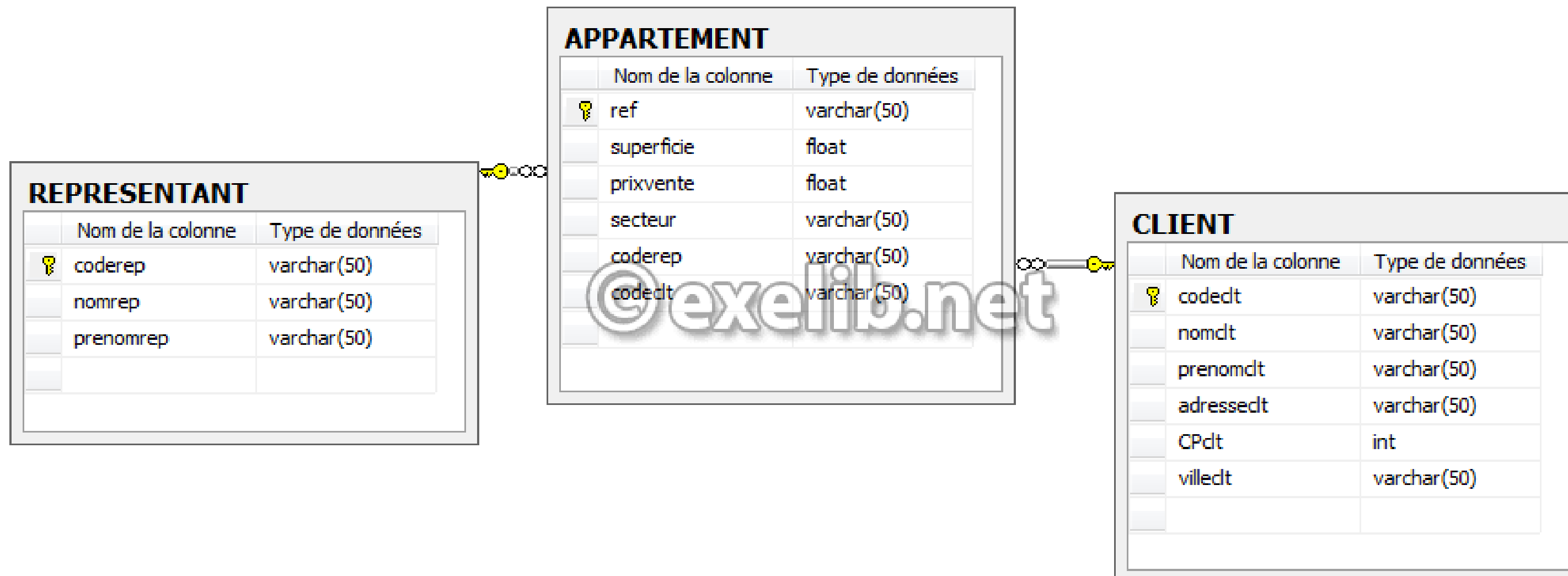

Récapitulatif d'ordre du SELECT

SELECT [DISTINCT] liste des champs, calcul
FROM table
[WHERE formule de restriction]
[GROUP BY liste des champs]
[HAVING formule de restriction]
[ORDER BY liste des champs]

Exercice d'application n°5 : 1/2

Jointure de tables

- Créez la BDD Immobilier, contenant les tables et relations suivantes :





Exercice d'application n°5 : 2/2

Jointure de tables

Affichez les informations suivantes :

- la liste des clients classés par ordre alphabétique
- la moyenne par secteur des prix des appartements
- le nombre d'appartements par secteur pour les secteurs qui dépassent 10 appartements.
 - le nombre d'appartements dont la superficie est supérieure à 80 m² par secteur
- le prix max des appartements par secteur mais seulement qui dépassent 10 appartements.
 - la liste des clients et les appartements qu'ils ont loué.
 - la liste des appartements situés à Hivernage et gérés par Fadoua ALAMI
 - le nombre de clients venant de Fès par secteur.



Exercice d'application n°5 : Solution

Créer la BDD Immobilier

```
CREATE DATABASE Immobilier;

USE Immobilier;

CREATE TABLE Representant(
    coderep varchar(50) NOT null PRIMARY KEY,
    nomrep varchar(50),
    prenomrep varchar(50)
);

CREATE TABLE Appartement(
    ref varchar(50) NOT null PRIMARY KEY,
    superficie float,
    prixvente float,
    secteur varchar(50),
    coderep varchar(50),
    codeclt varchar(50)
);

CREATE TABLE Client(
    codeclt varchar(50) NOT null PRIMARY KEY,
    nomclt varchar(50),
    prenomclt varchar(50),
    adresseclt varchar(50),
    CPclt int,
    villeclt varchar(50)
);

ALTER TABLE Appartement
ADD CONSTRAINT FOREIGN KEY coderep REFERENCES Representant (coderep);

ALTER TABLE Appartement
ADD CONSTRAINT FOREIGN KEY codeclt REFERENCES Client (codeclt);
```



Exercice d'application n°5 : Solution

```
//Q1
select *
from CLIENT
order by nomclt, prenomclt
```

```
//Q2
select secteur, AVG(prixvente) as "Moyenne de prix de vente"
from APPARTEMENT
group by secteur
```

```
//Q3
select secteur, COUNT(ref) as "Nombre d'appartements"
from APPARTEMENT
group by secteur
having COUNT(ref) > 10
```

```
//Q4
select secteur, COUNT(ref) as "Nombre d'appartements"
from APPARTEMENT
where superficie > 80
group by secteur
```

```
//Q5
select secteur, MAX(prixvente) as "Prix maximal"
from APPARTEMENT
group by secteur
having COUNT(ref) > 10
```

```
//Q6
select c.*, a.ref, a.secteur, a.superficie, a.prixvente, a.coderep
from CLIENT c inner join APPARTEMENT a on c.codeclt = a.codeclt
```

```
//Q7
select a.*
from APPARTEMENT a inner join REPRESENTANT r on a.coderep = r.coderep
where a.secteur = 'Hivernage ' and r.nomrep = 'ALAMI' and r.prenomrep = 'Fadoua'
```

```
//Q8
select secteur, COUNT(c.codeclt) as "Nombre de clients"
from CLIENT c inner join APPARTEMENT a on c.codeclt = a.codeclt
where c.villeclt = 'Fes'
group by secteur
```


PHPMyAdmin

81

phpMyAdmin

Recent Favorites

- admin_cmsms
- admin_cmsms2
- admin_cmsmsms
- admin_ls2
- admin_ls3
- admin_ls4
- admin_ls5
- admin_ma1
- admin_ma2
- admin_ma3
- admin_mo1
- admin_mo2
- admin_mo3
- admin_mo4

Type to filter these, Enter to search X

1 >>>

New

- mo_analytics_indicator_calc
- mo_analytics_models
- mo_analytics_models_log
- mo_analytics_predictions
- mo_analytics_prediction_actio
- mo_analytics_predict_sample
- mo_analytics_train_samples
- mo_analytics_used_analysab
- mo_analytics_used_files
- mo_assign
- mo_assignfeedback_commer
- mo_assignfeedback_editpdf
- mo_assignfeedback_editpdf
- mo_assignfeedback_editpdf
- mo_assignfeedback_editpdf
- mo_assignfeedback_editpdf
- mo_assignfeedback_file
- mo_assignment

Server: localhost » Database: admin_mo4

Structure SQL Search Query Export Import Operations Routines Events Triggers More

Page number: 1 >>>

Filters

Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> mo_analytics_indicator_calc	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_analytics_models	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
<input type="checkbox"/> mo_analytics_models_log	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
<input type="checkbox"/> mo_analytics_predictions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
<input type="checkbox"/> mo_analytics_prediction_actions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
<input type="checkbox"/> mo_analytics_predict_samples	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_analytics_train_samples	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_analytics_used_analysables	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
<input type="checkbox"/> mo_analytics_used_files	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
<input type="checkbox"/> mo_assign	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignfeedback_comments	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignfeedback_editpdf_annot	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignfeedback_editpdf_cmnt	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignfeedback_editpdf_queue	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
<input type="checkbox"/> mo_assignfeedback_editpdf_quick	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
<input type="checkbox"/> mo_assignfeedback_editpdf_rot	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignfeedback_file	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignment	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
<input type="checkbox"/> mo_assignment_submissions	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	40.0 KiB	
<input type="checkbox"/> mo_assignment_upgrade	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	
<input type="checkbox"/> mo_assignmentsubmission_file	★ Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	24.0 KiB	

Console