



## SQL : Base de données

### Objectifs

- ✓ Modéliser, concevoir et administrer une base de données.
- ✓ Maîtriser le langage SQL pour réaliser des requêtes afin de manipuler ou intervenir sur les données.

Fichier(s) utile(s) pour ce cours :

 [employees.sql](#)

01

### A quoi sert une base de données ?

Une base de données (*en anglais : Database, abrégé DB*) est utile dans le cadre d'un projet informatique pour conserver des informations en mémoire.

A l'intérieur de la base de données, les informations sont classées, structurées et regroupées généralement par sujet.

Dans la grande majorité des cas une base de données est gérée par un logiciel moteur qui la manipule : un SGBD (Système de Gestion de Base de Données).

Lorsque nous déposons un message sur un forum et que 2 semaines plus tard le message est toujours présent. ce n'est pas magique, c'est naturellement parce que le message a été sauvegardé dans une base de données.

Les bases de données sont incontournables dans tous les domaines... et même hors du web.

Nous sommes tous enregistrés dans des Bases De Données sans forcément s'en préoccuper (ex: sécurité sociale, banque, etc.).

02

### Acronyme

**MCD** = Modèle Conceptuel de Données

Le modèle conceptuel de données nous permet de nous fournir un plan de notre base de données avant de la créer.

(Tout comme pour une maison : Avant de la construire, on établit un plan de construction).

**SGBD** = Système de Gestion de Base de Données

Le SGBD permet d'accueillir, d'exploiter et de faire fonctionner les bases de données (moteur).

Nous utiliserons Mysql (*d'autres SGBD existe : Oracle, Sql Server, PostgreSQL, etc.*).

**BDD** = Base De Données

La base de données représente l'emplacement des données sauvegardées

**SQL** = Structured Query Language

Le langage de requête SQL nous permet d'échanger des informations avec la base de données

Une fois que les informations ont été enregistrées, il est important de pouvoir les gérer (ajout, modification, suppression, consultation).

Toutes ces actions de gestion et manipulation passeront par une requête SQL.

03

### Contexte et utilisation d'une Base De Données Relationnelle

Avant de rentrer dans le vif du sujet il est bien important de comprendre à quoi sert le langage SQL sur les sites web pour que vous soyez intéressés pour le reste du cours.

- 1 - Exemple sur l'utilisation des bases de données avec un espace membre

Voici notre page d'inscription :

INSCRIPTION	
Pseudo	<input type="text" value="LaMarie"/>
Mot de passe	<input type="text" value="Azerty"/>
Nom	<input type="text" value="Plantier"/>
Prénom	<input type="text" value="Marie"/>
Date de Naissance	<input type="text" value="15"/> <input type="text" value="novembre"/> <input type="text" value="1991"/>
<input type="button" value="S'inscrire"/>	

Le code HTML nous permet de créer un formulaire d'inscription (on y déclare les balises `<form>` `<label>` `<input>`, etc).

Le code CSS va nous permettre de mettre en forme et styliser le formulaire (couleur, positionnement).

En revanche, nous ne pourrons pas récupérer les saisies à l'aide des langages HTML et CSS. En effet, HTML et CSS sont des langages uniquement de conception qui servent à concevoir la page web (et non pas engager des traitements sur les informations).

Nous aurons besoin d'un langage de programmation type PHP (d'autres langages existent pour récupérer des saisies d'un formulaire...).

Dans le cadre de cette page d'inscription, lorsque l'internaute cliquera sur le bouton inscription, nous devrons récupérer les saisies postées par l'internaute via le langage de traitement PHP.

Ensuite, nous allons formuler une requête SQL de type INSERTION pour enregistrer le compte du nouvel internaute dans la table membre de notre base de données.

Pourquoi enregistrons nous le membre ? et bien pour qu'on puisse retrouver sa trace s'il souhaite se connecter !

Généralement si un internaute fait la démarche de s'inscrire, c'est pour pouvoir se connecter au site web par la suite.

Voici notre table membre (dans notre base de données) :

idMembre	pseudo	motDePasse	nom	prenom	dateDeNaissance
1	juju	soleil	thierry	julien	1985-05-22
2	LaMarie	Azerty	Plantier	Marie	1991-11-15
3	joker	planete123	Cottier	Jonathan	1995-01-30

Une table est un emplacement dans la base de données relatif à un sujet précis.

Une table est toujours composée de colonnes (*aussi appelées champ*). Cela permet de classer les informations de manière structurée (nom, prénom, etc.)

*\* Les mots de passe ont été inscrits en clair pour la compréhension de cet exemple, ils sont généralement cryptés.*

Nous pouvons voir en position 2, la présence de LaMarie.

Le numéro idMembre s'incrémente seul (Auto-Increment).

Voici notre page de connexion :

Page de connexion

Pseudo

LaMarie

Mot de passe

Aezrty

Connexion

Lorsque l'internaute cliquera sur le bouton de connexion, il va falloir formuler une requête SQL qui vérifiera la présence du compte de l'internaute dans la base de données.

Nous aurons besoin de formuler une requête de SELECTION. Ceci nous permettra de comparer les informations que l'on reçoit (via le formulaire) avec les informations que nous avons déjà enregistrées en base de données.


Si les informations concordent, nous donnerons une sorte de « feu vert » pour valider la connexion. Sinon « feu rouge » l'accouplement pseudo / mot de passe ne sera pas bon.

Nous pouvons conclure par le fait qu'un espace membre ne peut pas être créé sans base de données. Sans BDD, pas d'espace membre fiable !

- 2 - Exemple sur l'utilisation des bases de données - Boutique

Voici un catalogue dans une boutique :

## Catalogue

<b>1</b> <b>Tshirt Jaune</b>  10 € <a href="#">ajout au panier</a>	<b>2</b> <b>Tshirt Bleu</b>  35 € <a href="#">ajout au panier</a>	<b>3</b> <b>Tshirt Rouge</b>  20 € <a href="#">ajout au panier</a>	<b>4</b> <b>Tshirt Gris</b>  25 € <a href="#">ajout au panier</a>	<b>5</b> <b>Tshirt Noir</b>  20 € <a href="#">ajout au panier</a>
<b>6</b> <b>Tshirt Blanc</b>  15 € <a href="#">ajout au panier</a>	<b>7</b> <b>Tshirt Vert</b>  25 € <a href="#">ajout au panier</a>			

Les produits ne sont pas écrits directement (*aussi appelé en « dur »*) dans le code HTML, il s'agit d'un affichage dynamique et donc les produits proviennent d'une base de données.

Pour afficher dans une page web les produits contenus d'une base de données il faut formuler une requête de SELECTION.

Nous allons cliquer sur le produit 3 tshirt rouge pour accéder à la fiche détaillée.

Voici la fiche produit du tshirt rouge :

### Fiche Produit



#### Tshirt Rouge

Ref: TSHIRT\_ROUGE

PROMO

NOUVEAUTE

▼ [Lire la description complète](#)

Couleur :

Taille :

Quantité :

En stock

-10%

**20 €**

Livraison offerte !\*

Ajouter au panier



Vous imaginez bien que si on considère qu'il y a 800 produits au sein de notre boutique, il n'y aura pas 800 fiches produits du type produit1.php, produit2.php, produit3.php, ..., produit800.php, c'est impensable ! la gestion serait impossible puisque cela évolue en permanence !

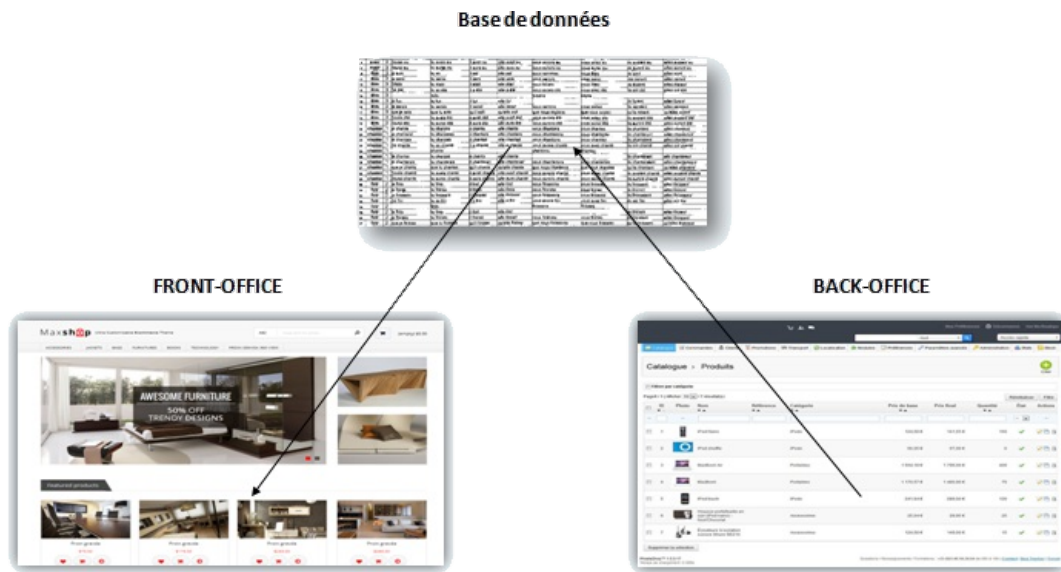
Il y aura qu'un seul fichier produit.php, un fichier modèle (template) qui pourra potentiellement accueillir les 800 produits.

Pour cela nous ne demanderons pas produit.php mais produit.php?id=1 ou produit.php?id=3, cela nous permettra de récupérer le numéro id et de formuler une requête SQL allant récupérer les informations sur le produit n° 3.

Conclusion : Sans base de données, pas de boutique ecommerce !

### - 3 - Exemple sur l'utilisation des bases de données - BackOffice

Voici un système de gestion de contenu (SGC) :



#### ➤ Le saviez-vous ?

La plupart du temps il n'y a pas 1 site web mais 2 !

➤ 1 site côté FRONT (c'est le site auquel tous les internautes ont accès)

➤ 1 site côté BACK (c'est l'interface de gestion pour le propriétaire / administrateur du site).

Lorsque le propriétaire / administrateur souhaite enregistrer un produit, écrire un article, effectuer un réglage, cela se passe du côté BackOffice et s'enregistre dans la base de données.

Lorsque l'internaute souhaite afficher le site (côté FrontOffice), cela récupère les informations (produits, articles, réglages, textes, ...) contenues à l'intérieur de la base de données.

Le BackOffice ne communique pas directement avec le FrontOffice, la base de données sert d'intermédiaire pour communiquer (c'est une sorte d'entrepôt où sont stockées les données).



#### ➤ Conclusion

Sans base de données pas d'interface BackOffice possible, pas de CMS non plus, car ils fonctionnent tous avec une BDD (*pas de wordpress, drupal, etc*).

Au vu de ces 3 exemples, vous comprendrez aisément que les bases de données et le langage SQL sont incontournables dans le monde du Web.

Auriez-vous des idées de site web AVEC base de données ?

- ecommerce
- site d'annonce
- site de mise en relation
- forum
- blog
- site vitrine d'informations
- etc.

Tous les sites, et même les sites vitrines ont besoin d'une base de données. (et oui les clients veulent être autonomes et s'assister d'une interface de gestion pour modifier les textes et images du site).

Auriez-vous des idées de site web SANS base de données ?

Est-ce que cela existe vraiment ? Oui certainement pour les anciens sites web, les sites statiques dont le contenu ne change jamais, ou encore les sites de présentation des développeurs eux mêmes, qui ont assez de connaissance et de patience pour pouvoir se permettre d'effectuer des modifications directement dans le code source. C'est une minorité absolue. Certainement moins de 5 %.

---

## Modélisation d'une base de données

---

04

### Modèle MCD

Avant de créer une base de données, il est essentiel de se poser et de réfléchir sur la modélisation.

En effet, une base de données va servir de support à une application informatique. C'est la raison pour laquelle il vaut mieux éviter de se tromper.

Rien est irréversible, mais changer la modélisation d'une base de données en cours de projet n'est pas recommandé car cela forcera certainement à réécrire une bonne partie du code (script) ce qui peut prendre énormément de temps et entraîner des conséquences telles qu'un dérèglement d'autres parties de l'application.

Comment modéliser une base de données ?

Nous pouvons modéliser sur papier si celle-ci n'est pas complexe mais dans la plupart des cas nous aurons besoin d'un logiciel pour nous accompagner et avoir une vue d'ensemble.

Un logiciel connu permet de modéliser une base de données : [Mysql WorkBench](#)

D'autres outils et logiciels performant existent. Pour une bonne conception de sa base de données, il faut réfléchir en terme de sujet en modélisant ceux du monde réel.

La modélisation se compose de différentes tables (table = sujet).

Par exemple, si nous vendons des produits sur notre site, nous aurons 1 table produit, mais aussi 1 table commande.

05

### Les tables

1 sujet représente 1 table dans une base de données. 1 table est un emplacement de sauvegarde.

Les données sont stockées à l'intérieur de tables. Une table peut être comparée à une liste, qui contient des enregistrements relatifs à un sujet bien défini.

Il faut réfléchir au sujet et à ses conséquences. Si nous vendons des produits, nous aurons certainement des membres. 1 table membre sera donc nécessaire.

En terme de fonctionnalité, si nous souhaitons proposer un abonnement newsletter aux membres, nous aurons besoin d'1 table newsletter repertoriant les membres abonnés.

1 table = 1 sujet

06

## Les colonnes / champs

Chaque table possède généralement plusieurs champs (aussi appelés colonnes).

Les colonnes / champs représentent des caractéristiques relatives au sujet (la table).

Pour savoir, quelle colonne mettre dans quelle table, il faut se poser la question suivante : qu'est-ce qui pourrait décrire mon sujet ? quelles sont les informations sur mon sujet ?

Par exemple, la table membre aura les champs : pseudo, mot de passe, nom, prenom, adresse, etc.

Nous enregistrerons toutes les caractéristiques d'un membre.

Pour la table produit, nous retrouverons des champs comme : titre, catégorie, couleur, taille, poids, prix, etc.

Nous n'irons pas mettre le champ pseudo dans la table produit. Un produit n'a pas de pseudo ! C'est illogique.

De la même manière, nous n'irons pas mettre le champ prix dans la table membre. Un membre représente une personne et n'a pas de prix !

07

## Les types de colonnes / champs

3 grandes catégories de champs ressortent : les types numériques, chaînes de caractères (texte) et temporels (dates).

Voici généralement les plus utilisés :

Type de champ	Description	Contexte
VARCHAR	Chaîne (jusqu'à 256 caractères)	Nous pourrions choisir ce champ pour enregistrer un pseudo, un email, le titre d'un produit ou d'un article de blog, etc.
TEXT	Chaîne de caractère (illimité)	Nous pourrions choisir ce champ pour enregistrer le texte d'un article de blog
		Nous pourrions choisir ce champ pour enregistrer les numéros d'un champ servant d'identifiant, un prix, un code

INT	numérique	Nous pourrions choisir ce champ pour enregistrer les numéros d'un champ servant d'identifiant, un prix, un code postal, le nombre de produits en stock, etc.
DATE	Date	Nous pourrions choisir ce champ pour enregistrer la date d'enregistrement d'une commande, d'un article de blog, etc.

Il existe d'autres types de champs complémentaires et utiles.

08

## Les identifiants (Clé Primaire - PK Primary Key)

Les identifiants sont des champs (colonnes) un peu particuliers car ils ne décrivent pas le sujet (ce n'est pas une des caractéristiques du sujet) mais ce sont des colonnes (champs) systématiquement présent dans chaque table et ce en première position.

Chaque table possède une colonne (champ) identifiant. Nous appellerons cela une clé primaire (PK).

Par exemple, nous pourrions l'appeler "id" ou encore "idProduit" pour la table produit, et "idMembre" pour la table membre.

Il s'agit d'une liste numérotée permettant de différencier chaque enregistrement de manière unique.

Le produit "tshirt rouge" deviendra (par exemple) le produit n°396 "tshirt rouge".

Pour éviter de choisir le numéro et faire des erreurs nous demanderons à le générer automatiquement avec l'Auto\_Increment.

09

## Auto\_Increment

Auto\_Increment est une option permettant de générer un numéro unique dans une colonne (champ) de type clé primaire (identifiant).

Par conséquent, retenez bien que le 1er champ de chaque table sera systématiquement un "id" qui sera PK (Primary Key) et AI (Auto\_Increment).

10

## NULL / NOT NULL

Dans chaque champ, nous pourrions indiquer si nous acceptons les valeurs NULL ou non (NOT NULL).

NULL est un type de valeur en informatique évitant de laisser un champ vide si nous n'avons pas d'informations à y déposer.

11

## Les relations

Il arrive parfois que les sujets interagissent entre eux.

Exemple : 1 membre commande 1 produit (ou 1 produit est commandé par 1 membre), nous devons enregistrer l'information dans 1 table commande.



Autre exemple : 1 conducteur conduit 1 véhicule (ou 1 véhicule est conduit par 1 conducteur). il nous faudra une table permettant de préciser quel conducteur va avec quel véhicule.

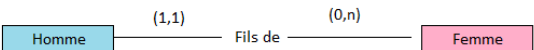
Pour cela, intéressons nous aux cardinalités !

12

Les cardinalités

Les cardinalités permettent de connaitre le chiffre minimum et maximum d'enregistrement pour une relation.

Exemple :



Dans notre exemple, un homme est le fils d'une femme et d'une seule (minimum = 1, maximum = 1).

En revanche, une femme peut avoir plusieurs enfants ou aucun (minimum = 0, maximum = n).

Autre exemple : 1 livre possède forcément 1 auteur et qu'1 seul. Relation (1,1). Minimum 1 auteur, Maximum 1 auteur.

1 auteur peut avoir écrit plusieurs livre (ou aucun). Relation (0,n). Minimum 0 livre, Maximum N livres

Pour ces raisons, nous pourrions imaginer les tables suivantes :

Auteur	
idAuteur	nom
1	auteur 1
2	auteur 2

Livre			
idLivre	titre	categorie	idAuteur
1	livre 1	romance	2
2	livre 2	drame	1

De cette manière nous savons : quel auteur à écrit quel livre.

Et aussi : quel livre a été écrit par quel auteur.

Au vue des cardinalités, nous n'avons pas besoin de créer une table de jointure spécifique et supplémentaire.

13

Table de jointure

Une table de jointure permet de faire le lien entre 2 tables.

Par exemple, prenons le cas d'une société taxis qui posséderait des conducteurs (table conducteur) et des véhicules (table véhicule).

Dans notre schéma, nous dirons :

➤ 1 conducteur peut conduire 0 ou plusieurs véhicules.

Relation (0,n). Minimum 0 véhicule, Maximum N véhicule.

➤ 1 véhicule peut être conduit par 0 ou plusieurs conducteur.

Relation (0,n). Minimum 0 conducteur, Maximum N conducteurs.

Comment savoir quel conducteur conduit quel véhicule ? ou dans l'autre sens : quel véhicule est conduit par quel conducteur ?

Une table de jointure nommée : conducteur\_vehicule (ou vehicule\_conducteur) sera donc créée avec les champs suivants :

- idVehiculeConducteur

- idConducteur

- idVehicule

De cette manière, nous sommes certains d'avoir de la visibilité sur les relations entre les conducteurs et les véhicules.

Conducteur	
idConducteur	nom
1	alexandre
2	julien

Vehicule	
idVehicule	modele
1	mercedes
2	bmw

Table de jointure : conducteur_vehicule		
idConducteurVehicule	idConducteur	idVehicule
1	1	2
2	2	2
3	1	1

Avec cette table de jointure, nous savons qu'Alexandre conduit à la fois la mercedes et la bmw tandis que julien conduit uniquement la bmw.

14

Clé étrangère

Les clés étrangères sont des champs forcément clé primaire dans leur table d'origine, se retrouvant également présent dans une table extérieure.  
Un champ clé étrangère (liste numérotée) placé à l'extérieur de leur table d'origine permet de faire la relation avec un autre sujet.

Exemple : le champ idConducteur dans la table conducteur est **Primary Key** mais se retrouve **Foreign Key** dans la table conducteur\_vehicule.

En tant que Primay Key (clé primaire), nous activerons toujours l'option `auto_increment`.

Lorsque ces champs sont FK - Foreign Key, ils ne posséderont pas l'option `auto_increment` (les n° de conducteurs en fonction des véhicules qu'ils conduisent seront totalement aléatoires, il ne s'agira pas de rendre unique un conducteur mais d'effectuer une relation entre un conducteur et un véhicule).

Table : conducteur		
champ	type	specificité
idConducteur	INT	PK - AI (Primary Key)
nom	VARCHAR	-

Table : vehicule		
champ	type	specificité
idVehicule	INT	PK - AI (Primary Key)
modele	VARCHAR	-

Table : conducteur_vehicule		
champ	type	specificité
idVehiculeConducteur	INT	PK - AI (Primary Key)
idConducteur	INT	FK (Foreign Key)
idVehicule	INT	FK (Foreign Key)

15

## Les Enregistrements

Chaque enregistrement représente 1 ligne dans la table de la base de données.

16

## Les Requêtes

Une requête permet de poser une question afin d'obtenir une réponse, ou encore de donner un ordre.

Comment formuler une requête SQL ?

Une requête se fait en 3 étapes :

Formulation -> Exécution -> Résultat(s).

Il faut d'abord savoir ce que l'on veut en français avant de le formuler en sql.

1 requete = 1 question ou 1 action

Il y a 4 grands types de requetes possibles :

➤ requete de selection

(requête question/réponse, nous faisons une demande via une question et obtenons une réponse)

➤ requete d'insertion

(requête d'action, impact sur les données)

➤ requete de modification

(requête d'action, impact sur les données)

➤ requete de suppression

(requête d'action, impact sur les données)

Nous terminerons toutes nos requêtes par un point-virgule pour que MYSQL sache que nous avons terminé d'écrire et qu'il doit exécuter notre demande.

## Requête de Selection

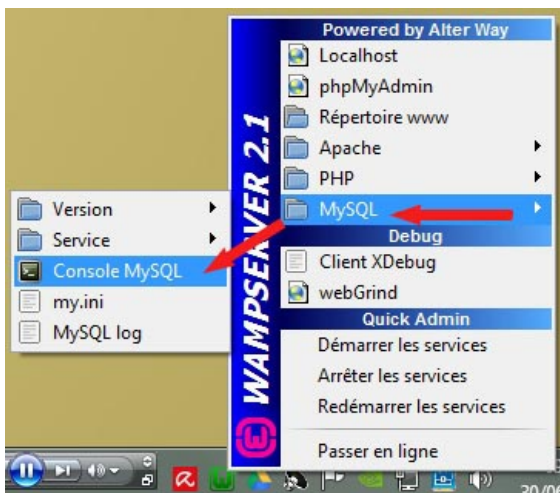
17

### Création d'une base de données

Nous pouvons créer une base de données à l'aide de la console Mysql ou du gestionnaire PhpMyAdmin.

Création de la Base de données :  entreprise

Avec la console Mysql :



➤ Cliquez sur l'icone de wamp, puis cliquer sur Mysql > Mysql Console.

Dans la console, vous pouvez cliquer sur l'icone en haut à gauche > propriété > configuration > Taille de la fenetre pour augmenter en largeur

Si vous souhaitez faire un copier/coller, sachez que le `ctrl+v` ne fonctionne pas sur certaines versions. Dans la console, il sera préférable d'effectuer un clic droit + coller.

A l'inverse, si vous voulez copier quelque chose de la console et le coller dans un fichier, il faudra également faire un clic droit + sélectionner, prendre la partie qui vous intéresse (avec le clic gauche de la souris) et ensuite aller dans votre fichier et coller normalement le contenu.

```
c:\wamp\bin\mysql\mysql5.5.8\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.8-log MySQL Community Server (GPL)

Copyright (c) 2000, 2010, 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> create database entreprise ;
```

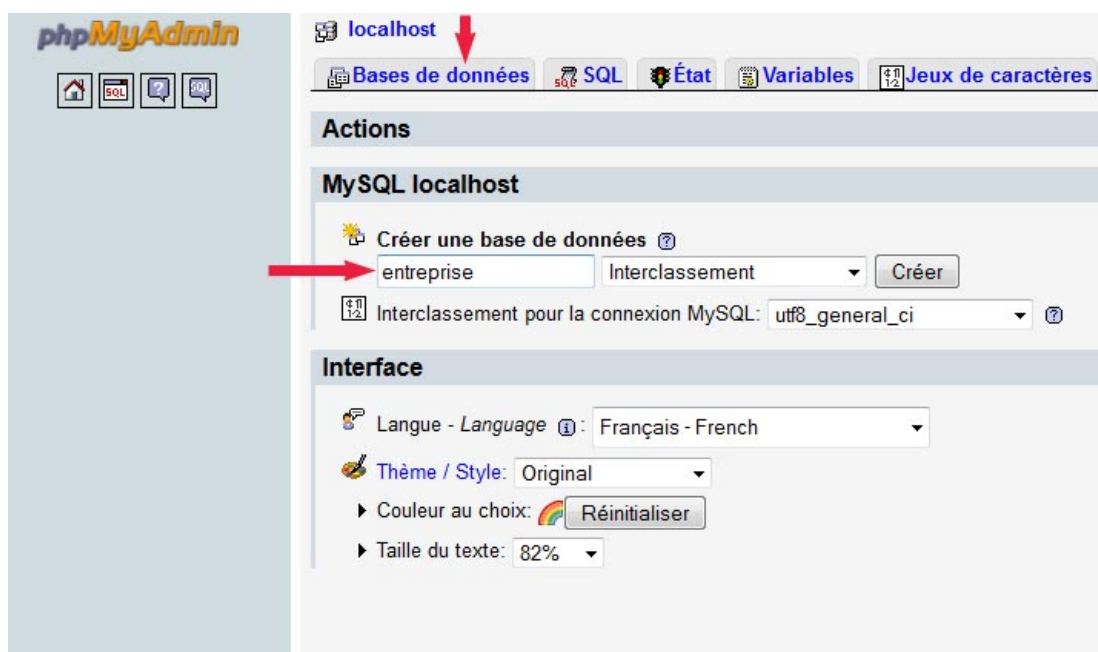
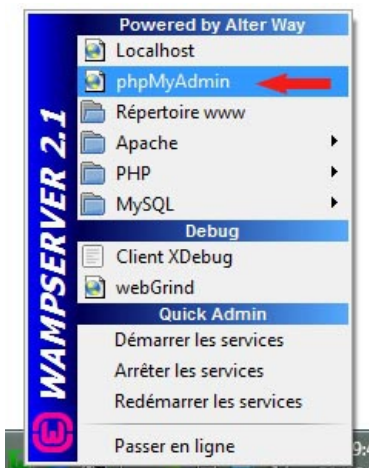
Saisir le code SQL suivant :

```
CREATE DATABASE entreprise ;
```

Si vous êtes en mode console, nous rajouterons une ligne de code pour dire au SGBD que nous souhaitons travailler sur notre base de données entreprise :

```
USE entreprise;
```

Avec le gestionnaire PhpMyAdmin :



Une fois la base de données "entreprise" créée, nous aurons besoin d'une table pour contenir des enregistrements :



Création de la table : **employees**

#### Base de données entreprise - Table employees

```
CREATE TABLE IF NOT EXISTS employees (
  id_employees int(3) NOT NULL AUTO_INCREMENT,
  prenom varchar(20) DEFAULT NULL,
  nom varchar(20) DEFAULT NULL,
  sexe enum('m','f') NOT NULL,
  service varchar(30) DEFAULT NULL,
  date_embauche date DEFAULT NULL,
  salaire float DEFAULT NULL,
  PRIMARY KEY (id_employees)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;
```

#### Insertion et enregistrement des informations :

Dans tous les cas, voici les enregistrements à insérer dans la base de données entreprise afin d'avoir du contenu sur lequel travailler :

```
INSERT INTO employees (id_employees, prenom, nom, sexe, service, date_embauche, salaire) VALUES
(350, 'Jean-pierre', 'Laborde', 'm', 'direction', '1999-12-09', 5000),
(388, 'Clement', 'Gallet', 'm', 'commercial', '2000-01-15', 2300),
(415, 'Thomas', 'Winter', 'm', 'commercial', '2000-05-03', 3550),
(417, 'Chloe', 'Dubar', 'f', 'production', '2001-09-05', 1900),
(491, 'Elodie', 'Fellier', 'f', 'secretariat', '2002-02-22', 1600),
(509, 'Fabrice', 'Grand', 'm', 'comptabilite', '2003-02-20', 1900),
(547, 'Melanie', 'Collier', 'f', 'commercial', '2004-09-08', 3100),
(592, 'Laura', 'Blanchet', 'f', 'direction', '2005-06-09', 4500),
(627, 'Guillaume', 'Miller', 'm', 'commercial', '2006-07-02', 1900),
(655, 'Celine', 'Perrin', 'f', 'commercial', '2006-09-10', 2700),
(699, 'Julien', 'Cottet', 'm', 'secretariat', '2007-01-18', 1390),
(701, 'Mathieu', 'Vignal', 'm', 'informatique', '2008-12-03', 2000),
(739, 'Thierry', 'Desprez', 'm', 'secretariat', '2009-11-17', 1500),
(780, 'Amandine', 'Thoyer', 'f', 'communication', '2010-01-23', 1500),
(802, 'Damien', 'Durand', 'm', 'informatique', '2010-07-05', 2250),
(854, 'Daniel', 'Chevel', 'm', 'informatique', '2011-09-28', 1700),
(876, 'Nathalie', 'Martin', 'f', 'juridique', '2012-01-12', 3200),
(900, 'Benoit', 'Lagarde', 'm', 'production', '2013-01-03', 2550),
(933, 'Emilie', 'Sennard', 'f', 'commercial', '2014-09-11', 1800),
(990, 'Stephanie', 'Lafaye', 'f', 'assistant', '2015-06-02', 1775);
```

Si on récapitule et qu'on développe :

#### Créer une base de données

```
CREATE DATABASE [nomdelabase] ;
```

#### Utiliser une base de données

```
USE [nomdelabase] ;
```

#### Lister les base de données

```
SHOW DATABASES;
```

#### Supprimer une base de données

```
DROP DATABASE [nomdelabase] ;
```

Supprimer une table d'une base de données

```
DROP TABLE [nomdelatable] ;
```

Vider le contenu d'une table d'une base de données

```
TRUNCATE [nomdelatable] ;
```

Observer la structure d'une table

```
DESC [nomdelatable] ;
```

Par exemple nous pourrions écrire :

```
DESC employes;
```

Cela nous permet de voir de quelle colonne est composée notre table.

19

Les requêtes de SELECTION

Afficher les noms et prénoms des employés travaillant dans l'entreprise

Requête SQL SELECT

```
SELECT nom, prenom FROM employes ;
```

Résultat

prenom	nom
Jean-pierre	Laborde
Clement	Gallet
Thomas	Winter
Chloe	Dubar
Elodie	Fellier
Fabrice	Grand
Melanie	Collier
Laura	Blanchet
Guillaume	Miller
Celine	Perrin
Julien	Cottet
Mathieu	Vignal
Thierry	Desprez
Amandine	Thoyer
Damien	Durand
Daniel	Chevel

Nathalie	Martin
Benoit	Lagarde
Emilie	Sennard
Stephanie	Lafaye

Afficher les services occupés dans l'entreprise

Requête SQL SELECT

SELECT service FROM employes ;

Résultat

service
direction
commercial
commercial
production
secretariat
comptabilite
commercial
direction
commercial
commercial
secretariat
informatique
secretariat
communication
informatique
informatique
juridique
production
commercial
assistant

Explications

Nous obtenons la liste des différents services dans l'entreprise. Ceci comporte des doublons car Mysql a renvoyé la colonne nommée service sans se préoccuper des doublons.

Afficher les services occupés dans l'entreprise (en évitant les doublons cette fois-ci)



## Requête SQL SELECT

```
SELECT DISTINCT(service) FROM employes ;
```



## Résultat

service
direction
commercial
production
secretariat
comptabilite
informatique
communication
juridique
assistant

## Explications

Le mot clé **DISTINCT** permet d'éviter les doublons.



## ➤ Bon à savoir

La fleche du haut (au clavier) est un raccourci pour l'historique des requêtes.

Les requêtes ne sont pas sensibles à la casse, mais une convention indique qu'il faut mettre les mots-clés des requetes en majuscule. Afficher la table employes (toutes les colonnes / champs)

## Requête SQL SELECT

```
SELECT id_employes, prenom, nom, sexe, service, date_embauche, salaire FROM employes;
```



## Résultat

id_employes	prenom	nom	sexe	service	date_embauche	salaire
350	Jean-pierre	Laborde	m	direction	1999-12-09	5000
388	Clement	Gallet	m	commercial	2000-01-15	2300
415	Thomas	Winter	m	commercial	2000-05-03	3550
417	Chloe	Dubar	f	production	2001-09-05	1900
491	Elodie	Fellier	f	secretariat	2002-02-22	1600
509	Fabrice	Grand	m	comptabilite	2003-02-20	1900
547	Melanie	Collier	f	commercial	2004-09-08	3100
592	Laura	Blanchet	f	direction	2005-06-09	4500
627	Guillaume	Miller	m	commercial	2006-07-02	1900
655	Celine	Perrin	f	commercial	2006-09-10	2700
699	Julien	Cottet	m	secretariat	2007-01-18	1390

701	Mathieu	Vignal	m	informatique	2008-12-03	2000
739	Thierry	Desprez	m	secretariat	2009-11-17	1500
780	Amandine	Thoyer	f	communication	2010-01-23	1500
802	Damien	Durand	m	informatique	2010-07-05	2250
854	Daniel	Chevel	m	informatique	2011-09-28	1700
876	Nathalie	Martin	f	juridique	2012-01-12	3200
900	Benoit	Lagarde	m	production	2013-01-03	2550
933	Emilie	Sennard	f	commercial	2014-09-11	1800
990	Stephanie	Lafaye	f	assistant	2015-06-02	1775

#### Explications

Nous selectionnons toutes les colonnes que nous souhaitons voir apparaitre dans le résultat.

#### Le raccourci \*

Afficher la table employes (toutes les colonnes / champs) avec le raccourci étoile " \* "

Requête SQL SELECT
SELECT * FROM employes;

#### Résultat

id_employes	prenom	nom	sexe	service	date_embauche	salaire
350	Jean-pierre	Laborde	m	direction	1999-12-09	5000
388	Clement	Gallet	m	commercial	2000-01-15	2300
415	Thomas	Winter	m	commercial	2000-05-03	3550
417	Chloe	Dubar	f	production	2001-09-05	1900
491	Elodie	Fellier	f	secretariat	2002-02-22	1600
509	Fabrice	Grand	m	comptabilite	2003-02-20	1900
547	Melanie	Collier	f	commercial	2004-09-08	3100
592	Laura	Blanchet	f	direction	2005-06-09	4500
627	Guillaume	Miller	m	commercial	2006-07-02	1900
655	Celine	Perrin	f	commercial	2006-09-10	2700
699	Julien	Cottet	m	secretariat	2007-01-18	1390
701	Mathieu	Vignal	m	informatique	2008-12-03	2000
739	Thierry	Desprez	m	secretariat	2009-11-17	1500
780	Amandine	Thoyer	f	communication	2010-01-23	1500
802	Damien	Durand	m	informatique	2010-07-05	2250
854	Daniel	Chevel	m	informatique	2011-09-28	1700
876	Nathalie	Martin	f	juridique	2012-01-12	3200
900	Benoit	Lagarde	m	production	2013-01-03	2550
933	Emilie	Sennard	f	commercial	2014-09-11	1800
990	Stephanie	Lafaye	f	assistant	2015-06-02	1775

## Explications

Le raccourci étoile permet de cibler toutes les colonnes afin de les afficher.

C'est une requête à connaître par coeur `SELECT * FROM [nomdelatable]`.

21

## Les conditions : Le mot clé WHERE

Il arrive parfois que nous ne souhaitons pas ressortir la totalité de nos données mais seulement une partie.

Pour cela nous allons utiliser une condition via la requête sql.

Afficher les employes (nom et prénom) du service informatique (uniquement)

### Requête SQL SELECT

```
SELECT nom, prenom FROM employes WHERE service='informatique';
```

### Résultat

nom	prenom
Vignal	Mathieu
Durand	Damien
Chevel	Daniel



#### ➤ Informations

Les apostrophes et quotes sont équivalentes (where service = 'informatique' pareil que where service = "informatique"). Nous ne mettrons pas de quotes ou d'apostrophes lorsque nous devons énoncer un chiffre (cela fonctionnerait quand même mais serait moins optimisé et contre nature car mysql devra convertir une chaine de caractères en integer).

## Explications

Dans notre contexte, le mot clé **WHERE** peut être traduit par "à condition que".

*"A condition que le service soit Informatique"*

Dans notre condition, nous respecterons toujours le format suivant : champ = valeur

Le champ sur lequel on annonce la condition doit être énoncé suivi de la valeur à transmettre.

Voici quelques explications supplémentaires :

Schema permettant d'expliquer et de traduire chaque mot SQL en FRANCAIS afin de décomposer une requête et de faciliter la compréhension

<u>SELECT</u>	<u>nom, prenom</u>	<u>FROM</u>	<u>employes</u>	<u>WHERE</u>	<u>service='informatique';</u>
Affiche moi	quoi ? nom des champs...	de ?	nom de la table...	à condition que ... Ce mot veut dire "où" dans ce contexte, nous pourrons le remplacer "à condition que"	...à condition que... que les employés fassent parti du service informatique champ = valeur Une condition se fait toujours sur un champ.

Afficher les employes ayant été recrutés entre 2006 et 2010

#### Requête SQL SELECT

```
SELECT nom, prenom, date_embauche FROM employes WHERE date_embauche BETWEEN '2006-01-01' AND '2010-12-31';
```

#### Résultat

nom	prenom	date_embauche
Miller	Guillaume	2006-07-02
Perrin	Celine	2006-09-10
Cottet	Julien	2007-01-18
Vignal	Mathieu	2008-12-03
Desprez	Thierry	2009-11-17
Thoyer	Amandine	2010-01-23
Durand	Damien	2010-07-05

#### Explications

Dans cette requête nous mettrons une condition sur le champ date\_embauche afin de selectionner uniquement les employés répondant aux critères énoncé dans la condition : avoir été recruté entre le 1er janvier 2006 et le 31 décembre 2010.

La date s'inscrit au format Américain : ANNEE - MOIS - JOUR

Imaginons que l'on veuille afficher les employés recrutés entre 2006 et aujourd'hui, aujourd'hui est une valeur qui change toutes les 24h et vous imaginez bien qu'on ne pourra pas mettre à jour constamment la date contenue dans la requête.

Pour cette raison nous allons utiliser une fonction prédéfinie CURDATE().

```
SELECT CURDATE();
```

#### Résultat

2016-04-10

Les fonctions prédéfinies permettent de réaliser un traitement (en l'occurrence, afficher la date du jour), elles sont toujours suivies de parenthèses.

Afficher les employes ayant été recrutés entre 2006 et aujourd'hui

#### Requête SQL SELECT

```
SELECT nom, prenom, date_embauche FROM employes WHERE date_embauche BETWEEN '2006-01-01' AND CURDATE();
```

#### Résultat

nom	prenom	date_embauche
Miller	Guillaume	2006-07-02

Perrin	Celine	2006-09-10
Cottet	Julien	2007-01-18
Vignal	Mathieu	2008-12-03
Desprez	Thierry	2009-11-17
Thoyer	Amandine	2010-01-23
Durand	Damien	2010-07-05
Chevel	Daniel	2011-09-28
Martin	Nathalie	2012-01-12
Lagarde	Benoit	2013-01-03
Sennard	Emilie	2014-09-11
Lafaye	Stephanie	2015-06-02

#### Explications

CURDATE() permet de sortir la date du jour (évolutive dans le temps).

## 23 Le mot clé LIKE

Like, très utilisé dans les moteurs de recherche, nous permet de trouver des enregistrements sans avoir d'informations précises mais seulement un mot ou une lettre pouvant coïncider avec les enregistrements en question.

Afficher les employés ayant un prénom commençant par la lettre "S"

```
Requête SQL SELECT
SELECT prenom FROM employes WHERE prenom LIKE 's%'
```

#### Résultat

prenom
Stephanie

#### Explications

LIKE nous permet d'annoncer une valeur approchante sans avoir pour autant la valeur exacte.

Le signe % nous permet d'annoncer une suite de caractères quelconques.

Dans notre cas, 's%' veut dire qui commence par la lettre S.

Nous aurions également pu inscrire '%s' pour dire qui termine par la lettre S.

Exemple :

```
Requête SQL SELECT
SELECT prenom FROM employes WHERE prenom LIKE '%s'
```

Résultat

prenom
Thomas

Explications

Le % étant placé après la lettre S, nous demandons au système de sortir tous les prénoms ayant une lettre S à la fin.

Autre exemple :

Requête SQL SELECT

SELECT prenom FROM employes WHERE prenom LIKE 'S-%'

Résultat

prenom
Jean-pierre

Explications

Avec la présence des signes pourcentages avant et après la lettre ou le mot recherché, nous demandons au système de trouver des enregistrements contenant l'expression recherchée.

Dans notre cas, nous isolerons les prénoms composés (ayant un trait d'union).

A quoi cela peut-il servir ?

Imaginons une table d'appartements (destinés à la location) avec les données suivantes :

id	ville	cp	adresse	prix	superficie
1	Paris	75015	rue A	730	25
2	Paris	75011	rue B	800	28
3	Paris	69003	rue C	820	30
4	Paris	75016	rue D	710	18
5	Paris	75008	rue E	920	33
6	Paris	75007	rue F	890	37

Voici une condition avec le signe égal (classique) :

Requête SQL SELECT

SELECT \* FROM appartement WHERE cp = 75;

Aucun enregistrements ne s'affichera suite à cette requête car aucun appartement n'est dans le département 75 mais dans le 75015, 75011, etc.

En revanche, avec la présence du mot clé like nous allons pouvoir cibler tous les appartements dont le code postal commence par 75 :

## Requête SQL SELECT

```
SELECT * FROM appartement WHERE cp LIKE '75%';
```

N'ayant pas réellement cette table de données sous la main, nous ne saisisons pas cette requête mais nous pouvons imaginer les résultats.

24

## Operateurs de comparaisons

- = "est égal"
- > "strictement supérieur"
- < "strictement inférieur"
- >= "supérieur ou égal"
- <= "inférieur ou égal"
- <> ou != "est différent"

## L'opérateur !=

Afficher tous les employés sauf ceux du service informatique

## Requête SQL SELECT

```
SELECT nom, prenom, service FROM employes WHERE service != 'informatique';
```

## Résultat

nom	prenom	service
Laborde	Jean-pierre	direction
Gallet	Clement	commercial
Winter	Thomas	commercial
Dubar	Chloe	production
Fellier	Elodie	secretariat
Grand	Fabrice	comptabilite
Collier	Melanie	commercial
Blanchet	Laura	direction
Miller	Guillaume	commercial
Perrin	Celine	commercial
Cottet	Julien	secretariat
Desprez	Thierry	secretariat
Thoyer	Amandine	communication
Martin	Nathalie	juridique
Lagarde	Benoit	production
Sennard	Emilie	commercial
Lafaye	Stephanie	assistant

Explications

Nous pouvons apercevoir dans la liste de résultats tous les employés sauf ceux du service informatique.

L'opérateur >

Afficher tous les employés gagnant un salaire supérieur à 3 000 €

Requête SQL SELECT

```
SELECT nom, prenom, service, salaire FROM employes WHERE salaire > 3000;
```

Résultat

nom	prenom	service	salaire
Laborde	Jean-pierre	direction	5000
Winter	Thomas	commercial	3550
Collier	Melanie	commercial	3100
Blanchet	Laura	direction	4500
Martin	Nathalie	juridique	3200

Explications

Nous isolons une portion de résultats : seulement les employés gagnant + de 3 000 €.

25

Les classements avec ORDER BY

Afficher tous les employés par ordre alphabétique (colonne prenom)

Requête SQL SELECT

```
SELECT prenom, nom FROM employes ORDER BY prenom ;
SELECT prenom, nom FROM employes ORDER BY prenom ASC ;
```

Avec ou sans le ASC la requête est la même. ASC est le mode de classement appliqué par défaut.  
ASC veut dire ASCendant croissant (du plus petit au plus grand).

Résultat

prenom	nom
Amandine	Thoyer
Benoît	Lagarde
Celine	Perrin
Chloe	Dubar
Clement	Gallet
Damien	Durand
Daniel	Chevel



Elodie	Fellier
Emilie	Sennard
Fabrice	Grand
Guillaume	Miller
Jean-pierre	Laborde
Julien	Cottet
Laura	Blanchet
Mathieu	Vignal
Melanie	Collier
Nathalie	Martin
Stephanie	Lafaye
Thierry	Desprez
Thomas	Winter

Explications

Comme vous pouvez le constater la liste des prénoms est bien classée dans l'ordre "Amandine" avant "Benoit" avant "Celine", etc.

Nous pouvons également demander le classement inverse avec le mot clé DESC :

Requête SQL SELECT

```
SELECT prenom, nom FROM employes ORDER BY prenom DESC ;
```

Résultat

prenom	nom
Thomas	Winter
Thierry	Desprez
Stephanie	Lafaye
Nathalie	Martin
Melanie	Collier
Mathieu	Vignal
Laura	Blanchet
Julien	Cottet
Jean-pierre	Laborde
Guillaume	Miller
Fabrice	Grand
Emilie	Sennard
Elodie	Fellier
Daniel	Chevel
Damien	Durand
Clement	Gallet
Chloe	Dubar
Celine	Perrin

Benoit	Lagarde
Amandine	Thoyer

### Explications

DESC permet d'annoncer DESCendant décroissant (du plus grand au plus petit).

Nous pouvons également réaliser plusieurs classements sur des champs/colonnes différents :

Requête SQL SELECT
SELECT nom, prenom, service, salaire FROM employes ORDER BY salaire ASC, prenom ASC;

### Résultat

nom	prenom	service	salaire
Cottet	Julien	secretariat	1390
Thoyer	Amandine	communication	1500
Desprez	Thierry	secretariat	1500
Fellier	Elodie	secretariat	1600
Chevel	Daniel	informatique	1700
Lafaye	Stephanie	assistant	1775
Sennard	Emilie	commercial	1800
Dubar	Chloe	production	1900
Grand	Fabrice	comptabilite	1900
Miller	Guillaume	commercial	1900
Vignal	Mathieu	informatique	2000
Durand	Damien	informatique	2250
Gallet	Clement	commercial	2300
Lagarde	Benoit	production	2550
Perrin	Celine	commercial	2700
Collier	Melanie	commercial	3100
Martin	Nathalie	juridique	3200
Winter	Thomas	commercial	3550
Blanchet	Laura	direction	4500
Laborde	Jean-pierre	direction	5000

### Explications

Le classement se fera sur le champ salaire, si jamais 2 employés gagnent le même salaire nous avons prévu un classement secondaire sur le champ prenom.

Pour 2 personnes qui gagneraient 1900 €, Fabrice apparaîtra avant Guillaume (F se trouve avant G dans l'alphabet).

Lorsque nous demandons d'afficher les produits des moins chers aux plus chers sur un site web ecommerce, la requête SQL utilise une clause ORDER BY pour faire le classement et vous proposer un affichage concordant.

ORDER BY sera donc précieux pour réaliser des classements sur vos prochains sites web.

Afficher les employés en les classant par salaire (du plus grand au plus petit) et en les affichant par 3.

#### Requête SQL SELECT

```
SELECT nom, prenom, service, salaire FROM employees ORDER BY salaire DESC LIMIT 0,3;
```

#### Résultat

nom	prenom	service	salaire
Laborde	Jean-pierre	direction	5000
Blanchet	Laura	direction	4500
Winter	Thomas	commercial	3550

#### Explications

LIMIT permet de limiter les résultats.

Le premier chiffre précise l'enregistrement de départ (0 définit le 1er enregistrement en informatique), le second chiffre nous indique le nombre de résultats que l'on souhaite obtenir.

Les prochains résultats seront affichés de cette manière :

#### Requête SQL SELECT

```
SELECT nom, prenom, service, salaire FROM employees ORDER BY salaire DESC LIMIT 3,3;
```

Et ainsi de suite :

#### Requête SQL SELECT

```
SELECT nom, prenom, service, salaire FROM employees ORDER BY salaire DESC LIMIT 6,3;
```

Puis :

#### Requête SQL SELECT

```
SELECT nom, prenom, service, salaire FROM employees ORDER BY salaire DESC LIMIT 9,3;
```

Le premier chiffre reste la position de laquelle on part, le second chiffre annonce toujours le nombre d'enregistrements à afficher (dans notre cas, on souhaite les afficher 3 par 3, donc ce 2e chiffre n'évolue pas).

Lorsque nous consultons un catalogue avec des centaines de produits, le site web ne peut pas tous les afficher d'un coup sinon la page web mettrait trop de temps à charger et la scrollbar serait trop longue, la requête SQL crée une pagination (affichage par groupe de 10 produits par exemples) grâce à la clause LIMIT.

LIMIT sera donc précieux sur vos prochains sites web pour créer des affichages avec pagination.

Il est possible de réaliser des calculs dans nos requêtes SQL.

Afficher la liste des employés avec leur salaire annuel (nous ferons un x12 pour simplifier le calcul dans le cadre de ce cours).

#### Requête SQL SELECT

```
SELECT nom, prenom, salaire*12 FROM employes;
```

#### Résultat

nom	prenom	salaire*12
Laborde	Jean-pierre	60000
Gallet	Clement	27600
Winter	Thomas	42600
Dubar	Chloe	22800
Fellier	Elodie	19200
Grand	Fabrice	22800
Collier	Melanie	37200
Blanchet	Laura	54000
Miller	Guillaume	22800
Perrin	Celine	32400
Cottet	Julien	16680
Vignal	Mathieu	24000
Desprez	Thierry	18000
Thoyer	Amandine	18000
Durand	Damien	27000
Chevel	Daniel	20400
Martin	Nathalie	38400
Lagarde	Benoit	30600
Sennard	Emilie	21600
Lafaye	Stephanie	21300

#### Explications

Le système va multiplier par 12 toutes les valeurs qu'il trouvera dans la colonne salaire.

Reprenons la requête précédente :

Requête SQL SELECT

SELECT nom, prenom, salaire\*12 AS salaire\_annuel FROM employes;

Résultat

nom	prenom	salaire_annuel
Laborde	Jean-pierre	60000
Gallet	Clement	27600
Winter	Thomas	42600
Dubar	Chloe	22800
Fellier	Elodie	19200
Grand	Fabrice	22800
Collier	Melanie	37200
Blanchet	Laura	54000
Miller	Guillaume	22800
Perrin	Celine	32400
Cottet	Julien	16680
Vignal	Mathieu	24000
Desprez	Thierry	18000
Thoyer	Amandine	18000
Durand	Damien	27000
Chevel	Daniel	20400
Martin	Nathalie	38400
Lagarde	Benoit	30600
Sennard	Emilie	21600
Lafaye	Stephanie	21300

Explications

Le mot clé AS permet de définir un ALIAS

29

Calculer la somme d'une colonne avec SUM

Nous pourrions calculer la somme que l'on paye pour nos salariés en faisant la somme de la colonne salaire :

Requête SQL SELECT

SELECT SUM(salaire\*12) FROM employes;

Résultat

--

SUM(salaire*12)
577380

#### Explications

Même si d'un point de vue comptable, le calcul de la masse salariale ne s'effectue pas comme ceci, nous voyons à travers cet exemple le calcul permettant de faire la somme d'une colonne.

Ceci sera pratique sur un site web pour calculer le CA (Chiffre d'Affaires) d'une boutique ecommerce.

30

## Calculer une moyenne avec AVG

Nous pourrions calculer le salaire moyen gagné dans notre entreprise :

Requête SQL SELECT

```
SELECT AVG(salaire) FROM employes;
```

#### Résultat

AVG(salaire)
2405.75

#### Explications

AVG est une fonction prédéfinie prenant comme argument (entre parenthèse) le nom de la colonne sur laquelle nous souhaitons calculer une moyenne.

Le salaire moyen est de 2405.75 €.

Ceci sera pratique sur un site web pour calculer le prix d'achat moyen d'une boutique ecommerce.

31

## Arrondir avec ROUND

Nous pourrions arrondir le calcul précédent :

Requête SQL SELECT

```
SELECT ROUND(AVG(salaire)) FROM employes;
```

#### Résultat

ROUND(AVG(salaire))
2406

### Explications

ROUND est une fonction prédéfinie prenant comme argument (entre parenthèse) le nombre à arrondir :

Nous aurions également pu inscrire : ROUND(AVG(salaire),2) pour obtenir 2 chiffres après la virgule (mais c'est déjà ce que nous avons de base).

Ceci sera pratique sur un site web pour arrondir des prix de vente avec le calcul de TVA sur une boutique ecommerce.

32

## Compter avec COUNT

Nous pourrions compter le nombre de résultats :

### Requête SQL SELECT

```
SELECT COUNT(*) FROM employes WHERE sexe='f';
```

### Résultat

COUNT(*)
9

### Explications

Dans ce contexte, cela nous permet de compter le nombre de femmes dans l'entreprise

33

## Isoler une valeur minimum/maximum avec MIN/MAX

Voyons comment trouver le salaire le plus faible dans notre liste d'employés :

### Requête SQL SELECT

```
SELECT MIN(salaire) FROM employes;
```

### Résultat

MIN(salaire)
1390

### Explications

Le mot clé MIN permet d'isoler le nombre minimum dans une colonne/champ.

De la même manière le mot clé MAX existe pour mettre en évidence le nombre maximum dans une colonne/champ.



### ➤ Attention

Pour autant, nous ne pourrions pas demander au système l'identité de cette personne via cette requête :

#### Requête SQL SELECT

```
SELECT prenom, MIN(salaire) FROM employes;
```

[?](#)

#### Résultat

prenom	MIN(salaire)
Jean-pierre	1390

Il n'y a pas d'erreur de requête mais le résultat est erroné, ce n'est pas Jean-Pierre qui gagne 1 390 €

Le système nous donne le premier résultat de la liste accompagné du salaire le plus faible.

En effet, si nous demandons au système de calculer le salaire le plus faible, nous ne pouvons pas lui demander dans le même temps de qui il s'agit. il ne pourra pas combiner les 2 instructions pour mener à 1 résultat cohérent.

Pour isoler le salaire le plus faible et en même temps connaître l'identité de cette personne, nous devons passer par une requête imbriquée (une requête dans une autre) :

#### Requête SQL SELECT

```
SELECT prenom, salaire FROM employes WHERE salaire = (SELECT MIN(salaire) FROM employes);
```

[?](#)

Le système va d'abord exécuter la requête entre parenthèse pour connaître le résultat et l'utiliser comme source pour l'autre requête.

Si nous lisons la sous requête (requête entre parenthèse) nous tomberons sur 1390, nous pourrions donc simplifier notre lecture par la requête suivante :

#### Requête SQL SELECT

```
SELECT prenom, salaire FROM employes WHERE salaire = 1390;
```

[?](#)

Nous demandons quelle personne gagne un salaire de 1390 €, il s'agit de Julien Cottet.

#### Résultat

prenom	salaire
Julien	1390

34

## Condition IN avec plusieurs valeurs

Nous pouvons appliquer une même condition comportant plusieurs valeurs grâce au mot clé IN :

#### Requête SQL SELECT

```
SELECT prenom,service FROM employes WHERE service IN('comptabilite', 'informatique');
```

[?](#)

#### Résultat

--	--



prenom	service
Fabrice	comptabilite
Mathieu	informatique
Damien	informatique
Daniel	informatique

#### Explications

Cette requête nous permet d'afficher les informaticiens et comptables dans l'entreprise.

Pour résumé :

- = (égal) permet d'annoncer 1 seule valeur
- IN permet d'annoncer plusieurs valeurs

Nous pouvons également exclure plusieurs valeurs avec NOT IN :

Requête SQL SELECT	
SELECT prenom, service FROM employes WHERE service NOT IN('comptabilite', 'informatique');	

#### Résultat

prenom	service
Jean-pierre	direction
Clement	commercial
Thomas	commercial
Chloe	production
Elodie	secretariat
Melanie	commercial
Laura	direction
Guillaume	commercial
Celine	commercial
Julien	secretariat
Thierry	secretariat
Amandine	communication
Nathalie	juridique
Benoit	production
Emilie	commercial
Stephanie	assistant

#### Explications

Cette requête nous permet d'afficher tous les employés sauf les informaticiens et comptables de l'entreprise.

Pour résumé :

- != (égal) permet d'exclure 1 seule valeur
- NOT IN permet d'exclure plusieurs valeurs

Il est possible d'appliquer plusieurs conditions différentes au sein de la même requête :

## Requête SQL SELECT

```
SELECT prenom, nom, salaire, service FROM employes WHERE service='commercial' AND salaire <= 2000 ;
```

## Résultat

prenom	nom	salaire	service
Guillaume	Miller	1900	commercial
Emilie	Sennard	1800	commercial

Explications

Dans cette requête nous demanderons à ce que les employés fassent partie du service commercial et gagnent un salaire supérieur ou égal à 2000 €.

Lorsque nous avons plusieurs conditions mélangeant OR (ou) et AND (et) l'ordre naturel de lecture peut être modifié, exemple :

## Requête SQL SELECT

```
SELECT prenom, nom, service, salaire FROM employes WHERE service='production' AND salaire= 1900 OR salaire=2300 ;
```

## Résultat

prenom	nom	service	salaire
Clement	Gallet	commercial	2300
Chloe	Dubar	production	1900

Explications

Dans cette requête nous partons dans l'idée de demander l'affichage des employés du service production gagnant un salaire de 1900 € ou 2300 € précisément.

Notre résultat inclut Clément Gallet du service commercial gagnant également 2300 €. Nous ne souhaitons pas le voir apparaître dans la liste car nous aimerions obtenir uniquement des employés du service production (qui peuvent gagner 1900 ou 2300 €).

Pourquoi un enregistrement indésiré est présent dans la liste ? Lorsqu'il y a AND et OR au sein de la même requête, le système lit la condition de la droite vers la gauche et il verra d'abord `salaire= 1900 OR salaire=2300` et ensuite `service='production'`

Pour garantir l'ordre naturel de lecture (de la gauche vers la droite), nous utiliserons des parenthèses :

Requête SQL SELECT

```
SELECT prenom, nom, service, salaire FROM employes WHERE service='production' AND (salaire= 1900 OR salaire=2300) ;
```

De cette manière, nous sommes sûr d'obtenir le(s) bon(s) résultat(s) :

Résultat

prenom	nom	service	salaire
Chloe	Dubar	production	1900

37

Les regroupements avec GROUP BY

Si nous voulons connaître le nombre d'employés par service, il peut être utile de faire des regroupements, voici un exemple :

Requête SQL SELECT

```
SELECT service, COUNT(*) AS nombre FROM employes GROUP BY service;
```

Résultat

service	nombre
assistant	1
commercial	6
communication	1
comptabilite	1
direction	2
informatique	3
juridique	1
production	2
secretariat	3

Explications

Le mot clé COUNT permet de compter chaque ligne d'enregistrement (+1), tandis que GROUP BY permet de les grouper ensemble (tout les +1) en fonction du même service.

Nous obtenons bien le nombre d'employés par service.

Pour imposer une condition dans une requête comportant un GROUP BY, nous utiliserons le mot clé HAVING :

Requête SQL SELECT

```
SELECT service, COUNT(*) AS nombre FROM employes GROUP BY service HAVING COUNT(*) > 2;
```

Résultat

service	nombre
commercial	6
informatique	3
secretariat	3

## Requête d'insertion

38

### La requête INSERT

Si nous souhaitons insérer un enregistrement, nous utiliserons une requête INSERT, voici un premier modèle :

#### Requête SQL INSERT

```
INSERT INTO employes (prenom, nom, sexe, service, date_embauche, salaire) VALUES ('alexis', 'richy', 'm', 'informatique', '2011-12-28', 1800);
```

#### Résultat

Query OK, 1 row affected.

#### Explications

Dans le premier modèle, nous annonçons tous les champs et toutes les valeurs correspondantes (sauf id\_employes auto-incrémenté), cela permet de choisir dans quels champs nous souhaitons insérer.

voici un second modèle :

#### Requête SQL INSERT

```
INSERT INTO employes VALUES ("', 'alexis', 'richy', 'm', 'informatique', '2012-01-28', 1800, 10);
```

#### Explications

Dans le second modèle, nous annonçons uniquement les valeurs, toutes les valeurs doivent obligatoirement être présentes. L'id sera précisé par des quotes vides afin qu'il se génère automatiquement.

Vous pouvez vérifier la présence du nouvel employé par une rapide requête de SELECTION:

#### Requête SQL SELECT

```
SELECT * FROM employes
```

## Requête de modification

39

### La requête UPDATE

Si nous souhaitons modifier un enregistrement, nous utiliserons une requête UPDATE, voici deux modèles :

#### Requête SQL UPDATE

```
UPDATE employes SET salaire=1871 WHERE id_employes = 7699;  
UPDATE employes SET salaire=1872, service="autre" WHERE id_employes = 7699;
```

#### Résultat

Query OK, 1 row affected.

#### Explications

Le premier modèle permet de modifier une information sur un enregistrement, tandis que le second modèle permet de modifier plusieurs informations sur un enregistrement.



#### ➤ Attention

il est plus prudent de modifier un enregistrement en ciblant son id (l'id étant unique cela permet d'éviter des erreurs).

Vous pouvez vérifier le changement par une rapide requête de SELECTION:

#### Requête SQL SELECT

```
SELECT * FROM employes
```

## Requête de suppression

40

### La requête DELETE

Si nous souhaitons supprimer un enregistrement, nous utiliserons une requête DELETE, voici deux modèles :

#### Requête SQL DELETE

```
DELETE FROM employes WHERE id_employes = 388 ;  
DELETE FROM employes WHERE service = 'informatique' ;
```

#### Résultat

Query OK, 1 row affected.

#### Explications

Le premier modèle permet de supprimer un employé précis, tandis que le second modèle permet de supprimer tous les employés du service informatique .



#### ➤ Attention

il est plus prudent de supprimer un enregistrement en ciblant son id (l'id étant unique cela permet d'éviter des erreurs).

Vous pouvez vérifier la suppression par une rapide requête de SELECTION:

#### Requête SQL SELECT

```
SELECT * FROM employes
```

41

### Exercice : Requête de SELECTION

Pour vous entrainer, voici quelques questions qui vous permettront d'élaborer des requêtes et mener vers la réponse :

1. Afficher la profession de l'employé 547.
2. Afficher la date d'embauche de : Amandine.
3. Afficher le nombre de commerciaux.
4. Afficher le coût des commerciaux sur 1 année.
5. Afficher le salaire moyen par service.
6. Afficher le nombre de recrutements sur l'année 2010.
7. Augmenter le salaire pour chaque employé de 100€.
8. Afficher le nombre de services (différents).
9. Afficher les informations de l'employé du service commercial gagnant le salaire le plus élevé
10. Afficher l'employé ayant été embauché en dernier.

42

### Correction : Requête de SELECTION

J'espère que vous avez réussi à en faire le maximum seul ! voici la correction :

1. Afficher la profession de l'employé 547.

#### Requête SQL 1

```
SELECT service FROM employes WHERE id_employes=547;
```

?

résultat 1

service
commercial

Explications : Nous utilisons une condition WHERE sur le champ id (demande de départ) pour cibler uniquement un employé en particulier.

2. Afficher la date d'embauche de : Amandine.

#### Requête SQL 2

```
SELECT date_embauche FROM employes WHERE prenom='amandine';
```

?

résultat 2

date_embauche
2010-01-23

Explications : Nous utilisons une condition WHERE sur le champ prenom (demande de départ) pour cibler uniquement un employé en particulier.

3. Afficher le nombre de commerciaux.

#### Requête SQL 3

```
SELECT COUNT(*) as 'nombre' FROM employes WHERE service='commercial';
```

?

résultat 3

nombre
6

Explications : Nous utilisons COUNT pour compter le nombre de commerciaux, nous ne souhaitons pas savoir qui ils sont.

4. Afficher le coût des commerciaux sur 1 année.

#### Requête SQL 4

```
SELECT SUM(salaire*12) FROM employes WHERE service='commercial';
```

résultat 4

SUM(salaire*12)
184200

Explications : Nous utilisons SUM pour faire la somme de la colonne salaire (que nous multiplions par 12) à condition que les employés fassent partie du service commercial (demande de départ).

5. Afficher le salaire moyen par service.

#### Requête SQL 5

```
SELECT service, round(AVG( salaire )) FROM employes GROUP BY service;
```

résultat 5

service	round(AVG( salaire ))
assistant	1775
commercial	2558
communication	1500
comptabilite	1900
direction	4750
informatique	1983
juridique	3200
production	2225
secretariat	1497

Explications : Nous utilisons une AVG pour calculer la moyenne des salaire et ROUND pour arrondir cette moyenne. Group By permet d'effectuer un regroupement, indispensable dans notre cas.

6. Afficher le nombre de recrutements sur l'année 2010.

#### Requête SQL 6

```
SELECT COUNT(*) as 'nb de recrutement' FROM employes WHERE date_embauche BETWEEN '2010-01-01' AND '2010-12-31';  
SELECT COUNT(*) as 'nb de recrutement' FROM employes WHERE date_embauche LIKE '2010%';  
SELECT COUNT(*) as 'nb de recrutement' FROM employes WHERE date_embauche >= '2010-01-01' AND date_embauche <= '2010-12-31';
```

résultat 6

nb de recrutement
-------------------

**Explications :** Plusieurs méthodes permettent de consulter des enregistrements entre deux dates précises : BETWEEN, LIKE, Opérateur de comparaison, etc.

7. Augmenter le salaire pour chaque employé de 100€.

#### Requête SQL 7

```
UPDATE employes SET salaire=salaire+100;
```

résultat 7Query OK, 1 row affected.

8. Afficher le nombre de services (différents).

#### Requête SQL 8

```
SELECT COUNT(DISTINCT service) FROM employes;
```

résultat 8

COUNT(DISTINCT service)
9

**Explications :** Nous utilisons une condition WHERE sur le champ prenom (demande de départ) pour cibler uniquement un employé en particulier.

9. Afficher les informations de l'employé du service commercial gagnant le salaire le plus élevé

#### Requête SQL 9

```
SELECT prenom, salaire FROM employes WHERE service='commercial' AND salaire = ( SELECT MAX( salaire ) FROM employes WHERE service='commercial' );
```

résultat 9

prenom	salaire
Thomas	3550

**Explications :** Nous passerons par une requête imbriquée pour connaître le salaire maximum gagné par un employé du service commercial. Une fois ce salaire connu, nous l'utiliserons pour savoir qui le gagne dans la liste des employés. La sous-requête (entre parenthèse) est exécutée avant. Le système interprète d'abord la requête entre parenthèse avant de s'en servir pour le reste de la requête.

10. Afficher l'employé ayant été embauché en dernier.

#### Requête SQL 10


```
SELECT * FROM employes WHERE date_embauche = (SELECT max(date_embauche) FROM employes);
```

résultat 10

id_employes	prenom	nom	sexe	service	date_embauche	salaire
990	Stephanie	Lafaye	f	assistant	2015-06-02	1775

**Explications :** Même principe que la requête précédente, avec la requête imbriquée, nous isolons la date d'embauche la plus grande et regardons ensuite qui a été recruté à cette date.



Fichier(s) utile(s) pour ce cours :  [bibliotheque.sql](#)

43

### Requête Imbriquée

Pour la suite du cours, nous allons nous entrainer sur des requêtes imbriquées.

Afin d'avoir un cas concret et un contexte favorable à ce type de requête, nous allons modéliser la base de données d'une bibliothèque.

Comme nous sommes encore au début de notre entrainement, nous prendrons une version minimale (on ne va pas se rajouter trop de travail :p) :

Quels sont les sujets d'une bibliothèque ?

- Des livres
- Des abonnés (ou adhérent, ou client).

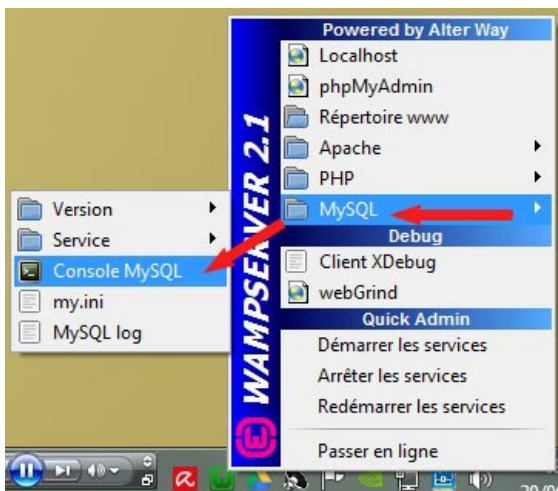
Et nous aurons également une table qui permet de savoir quel livre à été emprunté par quel abonné et à quelle date (en gros: qui a emprunté quoi et quand ?).

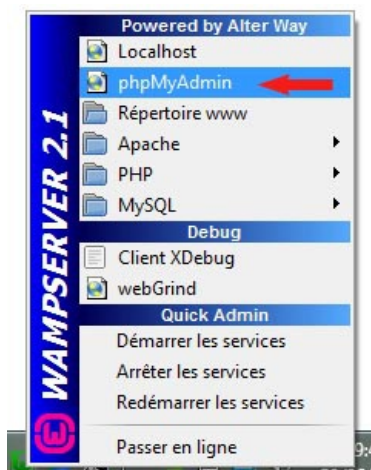
Nous appellerons cette table : **emprunt**.

Ceci nous permettra également de réaliser des requêtes sur plusieurs tables en même temps.

Création de la Base de données :  **bibliotheque**

Pour ce faire, vous pouvez utiliser la console mysql ou phpmyadmin :





L'objectif du prochain exercice sera de créer les tables suivantes :

Table : Abonne					
Field	Type	Null	Key	Default	Extra
id_abonne	int(3)	NO	PRI		auto_increment
prenom	varchar(20)	NO			

Explications : id\_abonne représente le champ clé primaire et auto increment (PK - AI), il s'agira d'une liste de chiffres uniques permettant de différencier chaque enregistrement.

Nous garderons en mémoire uniquement le prenom de chaque abonné mais nous pourrions enregistrer d'autres informations comme son nom, adresse, etc.

Table : Livre					
Field	Type	Null	Key	Default	Extra
id_livre	int(3)	NO	PRI		auto_increment
auteur	varchar(30)	NO			
titre	varchar(30)	NO			

Explications : id\_livre représente le champ clé primaire et auto increment (PK - AI), il s'agira d'une liste de chiffres uniques permettant de différencier chaque enregistrement.

Chaque livre possède un titre et a été écrit par un auteur (nous aurions pu également enregistrer d'autres informations).

Table : Emprunt					
Field	Type	Null	Key	Default	Extra
id_emprunt	int(3)	NO	PRI		auto_increment
id_livre	int(3)	YES			
id_abonne	int(3)	YES			
date_sortie	date	NO			
date_rendu	date	YES			

Explications : id\_emprunt représente le champ clé primaire et auto increment (PK - AI), il s'agira d'une liste de chiffres uniques permettant de différencier chaque enregistrement.

Dans la table emprunt, id\_livre sera un champ clé étrangère en référence au champ id\_livre de la table livre.

De la même manière, le champ id\_abonne de la table emprunt sera une clé étrangère au champ id\_abonne de la table abonne.

Pour effectuer la création, libre à vous d'utiliser PhpMyAdmin ou la Console Mysql.

45

## Correction : Création de tables

Voici le code SQL permettant de créer vos 3 tables :

### Création des tables abonne, livre et emprunt sur la base de données bibliotheque

```
CREATE DATABASE IF NOT EXISTS bibliotheque ;

USE bibliotheque ;

CREATE TABLE abonne (
  id_abonne INT(3) NOT NULL AUTO_INCREMENT,
  prenom VARCHAR(20) NOT NULL,
  PRIMARY KEY (id_abonne)
) ENGINE=InnoDB ;

CREATE TABLE emprunt (
  id_emprunt INT(3) NOT NULL AUTO_INCREMENT,
  id_livre INT(3) DEFAULT NULL,
  id_abonne INT(3) DEFAULT NULL,
  date_sortie DATE NOT NULL,
  date_rendu DATE DEFAULT NULL,
  PRIMARY KEY (id_emprunt)
) ENGINE=InnoDB ;

CREATE TABLE livre (
  id_livre INT(3) NOT NULL AUTO_INCREMENT,
  auteur VARCHAR(30) NOT NULL,
  titre VARCHAR(30) NOT NULL,
  PRIMARY KEY (id_livre)
) ENGINE=InnoDB ;
```

Nous pourrions ajouter les clés étrangères comme ceci :

### Création des tables abonne, livre et emprunt sur la base de données bibliotheque

```
ALTER TABLE emprunt ADD FOREIGN KEY ( id_livre ) REFERENCES bibliotheque.livre (id_livre);

ALTER TABLE emprunt ADD FOREIGN KEY ( id_abonne ) REFERENCES bibliotheque.abonne (id_abonne);
```

46

## Exercice : Insertion du contenu

Maintenant que nous avons nos 3 tables, il faut penser à insérer du contenu à l'intérieur.

Voici les données attendues :

abonne	
id_abonne	prenom
1	Guillaume
2	Benoit

3	Chloe
4	Laura

livre		
id_livre	auteur	titre
100	GUY DE MAUPASSANT	Une vie
101	GUY DE MAUPASSANT	Bel-Ami
102	HONORE DE BALZAC	Le père Goriot
103	ALPHONSE DAUDET	Le Petit chose
104	ALEXANDRE DUMAS	La Reine Margot
105	ALEXANDRE DUMAS	Les Trois Mousquetaires

id_emprunt	id_livre	id_abonne	date_sortie	date_rendu
1	100	1	2014-12-17	2014-12-18
2	101	2	2014-12-18	2014-12-20
3	100	3	2014-12-19	2014-12-22
4	103	4	2014-12-19	2014-12-22
5	104	1	2014-12-19	2014-12-28
6	105	2	2015-03-20	2015-03-26
7	105	3	2015-06-13	NULL
8	100	2	2015-06-15	NULL

\* Seule particularité : le champ date\_rendu de la table emprunt comportera la valeur NULL si le livre se trouve hors de la bibliothèque.

Pour effectuer les insertions, libre à vous d'utiliser PhpMyAdmin ou la Console Mysql.

Voici le code SQL permettant de créer vos enregistrements :

#### Insertion d'enregistrement sur les tables abonne, livre et emprunt, pour la base de données bibliotheque

```
INSERT INTO abonne (id_abonne, prenom) VALUES
(1, 'Guillaume'),
(2, 'Benoit'),
(3, 'Chloe'),
(4, 'Laura');

INSERT INTO livre (id_livre, auteur, titre) VALUES
(100, 'GUY DE MAUPASSANT', 'Une vie'),
(101, 'GUY DE MAUPASSANT', 'Bel-Ami '),
(102, 'HONORE DE BALZAC', 'Le père Goriot'),
(103, 'ALPHONSE DAUDET', 'Le Petit chose'),
(104, 'ALEXANDRE DUMAS', 'La Reine Margot'),
(105, 'ALEXANDRE DUMAS', 'Les Trois Mousquetaires');

INSERT INTO emprunt (id_emprunt, id_livre, id_abonne, date_sortie, date_rendu) VALUES
(1, 100, 1, '2014-12-17', '2014-12-18'),
(2, 101, 2, '2014-12-18', '2014-12-20'),
(3, 100, 3, '2014-12-19', '2014-12-22'),
(4, 103, 4, '2014-12-19', '2014-12-22'),
(5, 104, 1, '2014-12-19', '2014-12-28'),
```

```
(6, 105, 2, '2015-03-20', '2015-03-26'),  
(7, 105, 3, '2015-06-13', NULL),  
(8, 100, 2, '2015-06-15', NULL);
```

48

## Requête Imbriquée sur plusieurs tables

Avant d'effectuer des requêtes sur plusieurs tables, vous pouvez prendre connaissance de vos nouvelles données en effectuant les requêtes suivantes :

### Requête SQL SELECT

```
SELECT * FROM abonne ;  
SELECT * FROM livre ;  
SELECT * FROM emprunt ;
```

Pour une première requête, nous essaierons d'afficher les id des livres encore dans la nature n'ayant pas été rendus à la bibliothèque

La logique voudrait que l'on inscrive la requête suivante :

### Requête SQL SELECT

```
SELECT id_livre FROM emprunt WHERE date_rendu = NULL;
```

Cas particulier la valeur NULL se teste avec le mot clé IS, voici donc la bonne requête :

### Requête SQL SELECT

```
SELECT id_livre FROM emprunt WHERE date_rendu IS NULL;
```

#### Résultat

id_livre
105
100

Pour la prochaine requête, essayons de trouver les titres des livres dans la nature n'ayant pas été rendus à la bibliothèque

*Si nous souhaitons savoir si un livre a été rendu ou non, nous aurons besoin de la table emprunt (avec le champ date\_rendu).*

*Si nous souhaitons connaître le titre d'un livre, nous aurons besoin de la table livre (avec le champ titre).*

Nous allons donc avoir besoin d'exécuter une requête imbriquée sur 2 tables différentes.

**Demande :** Quels sont les titres des livres n'ayant pas été rendus à la bibliothèque ?

**Requête :**

### SQL - Requête Imbriquée - SELECT

```
SELECT titre FROM livre WHERE id_livre IN  
(SELECT id_livre FROM emprunt WHERE date_rendu IS NULL);
```

**Explications :** La requête entre parenthèse s'exécute en premier et nous sort les id de livre 105 et 100.

Ensuite, nous demandons les titres des livres dans la table emprunt correspondant aux id des livres 105 et 100.

Nous utilisons le mot clé IN et non pas le signe = (égal) car plusieurs résultats seront renvoyés par la sous requête.

Dans le cadre de cet exemple, pour assurer la jonction et la correspondance des données, nous utilisons le champ id\_livre, le champ annoncé en condition WHERE doit être le même que le champ SELECT juste à après.

Résultat

:

résultat

titre
Une vie
Les Trois Mousquetaires



➤ Attention

Pour qu'une requête imbriquée sur 2 tables soit possible, il faut absolument qu'une information soit commune aux 2 tables afin que la jonction et la correspondance entre les données puissent être faites.

Dans notre cas, nous avons utilisé le champ id\_livre pour joindre et faire correspondre les informations de la table livre et de la table emprunt.

**Demande** : Nous aimerions connaître le n° (id) de(s) livre(s) que Chloé a emprunté à la bibliothèque

**Indices** : Chloé se trouve dans la table abonné, les informations (id) sur les livres qu'elle a empruntés se trouvent dans la table emprunt.

Il est donc nécessaire de faire la passerelle entre ces deux tables.

Comment faire ? un champ se répète à la fois dans la table abonné et dans la table emprunt, il s'agit du champ id\_abonne. C'est grâce à ce champ que nous pourrons faire la passerelle entre les deux tables.

N'oubliez pas qu'un champ doit absolument être commun aux 2 tables pour que la jonction et la correspondance des données puissent se faire dans le cadre d'une requête imbriquée.

Requête :

SQL - Requête Imbriquée - SELECT

```
SELECT titre FROM livre WHERE id_livre IN  
(SELECT id_livre FROM emprunt WHERE date_rendu IS NULL);
```

**Explications** : La requête entre parenthèse s'exécute en premier et nous sort de la table emprunt les id de livre 105 et 100 (ayant une date de rendu à null et donc hors de la bibliothèque).

Ensuite, nous demandons les titres des livres (dans la table livre) correspondant aux id des livres 105 et 100.

Résultat

:

résultat

titre
Une vie
Les Trois Mousquetaires

**Demande** : Afficher les prénoms des abonnés ayant emprunté un livre le 19/12/2014

Indices :

- La date de sortie d'un livre se trouve dans la table emprunt.
- Les prénoms des abonnés se trouvent dans la table abonné.
- Le champ en commun entre ces deux tables permettant de faire la passerelle est le champ id\_abonné.

Requête :

#### SQL - Requête Imbriquée - SELECT

```
SELECT prenom FROM abonné WHERE id_abonné IN  
( SELECT id_abonné FROM emprunt WHERE date_sortie='2014-12-19' );
```

Explications : La requête entre parenthèse s'exécute en premier et nous sort de la table emprunt les id des abonnés ayant emprunté un livre le 19/12/2014.

Ces id sont 1, 3 et 4. Lorsqu'une sous requête est susceptible de sortir plusieurs résultats, il faut prévoir le mot clé IN acceptant plusieurs valeurs, et non pas le signe égal = n'acceptant qu'une seule valeur.

Ensuite, nous demandons les prénoms des abonnés correspondant aux n° id 1, 3 et 4.

Résultat

:

résultat

prenom
Guillaume
Chloe
Laura

Demande : Afficher la liste des abonnés ayant déjà emprunté un livre d'Alphonse DAUDET

Indices :

- La liste des abonnés se trouvent dans la table abonné.
- Les n° id des livres écrits par Alphonse Daudet se trouvent dans la table livre.
- La liste des emprunts (qui a emprunté quoi ?) se trouve dans la table emprunt.
- Nous ne pouvons pas relier la table abonné directement avec la table livre (car ces deux tables ne possèdent pas de champ en commun).
- Nous pouvons relier la table livre avec la table emprunt. Nous pouvons aussi relier la table abonné avec la table emprunt.

Requête :

#### SQL - Requête Imbriquée - SELECT

```
SELECT prenom FROM abonné WHERE id_abonné IN  
(SELECT id_abonné FROM emprunt WHERE id_livre IN  
(SELECT id_livre FROM livre WHERE auteur='ALPHONSE DAUDET')) ;
```

Explications : La dernière requête entre parenthèse s'exécute en premier et nous sort de la table livre les id des livres écrits par Alphonse Daudet.

*SELECT id\_livre FROM livre WHERE auteur='ALPHONSE DAUDET';*

Le résultat de cette requête est :

id_livre
103

Ensuite, dans la requête du milieu (entre parenthèses) nous demandons à obtenir les id d'abonnés des personnes ayant emprunté ces id de livres (le 103).

*SELECT id\_abonné FROM emprunt WHERE id\_livre IN(103);*

Le résultat de cette requête est :

id_abonne
4

Pour terminer, dans la requête du dessus, nous demandons à afficher les prénoms des abonnés répondant à ces id d'abonnés (le 4).

*SELECT prenom FROM abonne WHERE id\_abonne IN(4);*

Le résultat de cette requête est :

prenom
Laura

#### Résultat

: C'est donc le résultat final.

résultat

prenom
Laura

**Demande** : Afficher le(s) titre de(s) livre(s) que Chloé a emprunté à la bibliothèque.

**Indices** :

- Les titres des livres se trouvent dans la table livre.
- Les prénoms des abonnés se trouvent dans la table abonné.
- La liste des emprunts (qui a emprunté quoi ?) se trouve dans la table emprunt.
- Nous ne pouvons pas relier la table abonné directement avec la table livre (car ces deux tables ne possèdent pas de champ en commun).
- Nous pouvons relier la table livre avec la table emprunt. Nous pouvons aussi relier la table abonne avec la table emprunt.

**Requête** :

#### SQL - Requête Imbriquée - SELECT

```
SELECT titre FROM livre WHERE id_livre IN
(SELECT id_livre FROM emprunt WHERE id_abonne =
(SELECT id_abonne FROM abonne WHERE prenom='chloe') );
```

**Explications** : La dernière requête entre parenthèse s'exécute en premier et nous sort de la table abonné l'id d'abonné de Chloé (dans le cadre de cet entraînement, nous considérerons qu'il n'y a qu'une seule Chloé).

*SELECT id\_abonne FROM abonne WHERE prenom='chloe';*

Le résultat de cette requête est :

id_abonne
3

Ensuite, dans la requête du milieu (entre parenthèses) nous demandons à obtenir les id de livres ayant été emprunté par l'abonné n°3

*SELECT id\_livre FROM emprunt WHERE id\_abonne = 3;*

Le résultat de cette requête est :

id_livre
100
105



Pour terminer, dans la requête du dessus, nous demandons à afficher les titres des livres correspondants aux n° de livre 100 et 105..  
`SELECT titre FROM livre WHERE id_livre IN(100,105);`

Le résultat de cette requête est :

titre
Une vie
Les Trois Mousquetaires

**Résultat**  
: C'est donc le résultat final.

résultat

titre
Une vie
Les Trois Mousquetaires

**Demande** : Afficher le(s) titre de(s) livre(s) que Chloé n'a pas encore emprunté à la bibliothèque.

- Indices** :
- Les titres des livres se trouvent dans la table livre.
  - Les prénoms des abonnés se trouvent dans la table abonné.
  - La liste des emprunts (qui a emprunté quoi ?) se trouve dans la table emprunt.
  - Nous ne pouvons pas relier la table abonné directement avec la table livre (car ces deux tables ne possèdent pas de champ en commun).
  - Nous pouvons relier la table livre avec la table emprunt. Nous pouvons aussi relier la table abonne avec la table emprunt.

**Requête** :

SQL - Requête Imbriquée - SELECT

```
SELECT titre FROM livre WHERE id_livre NOT IN
  (SELECT id_livre FROM emprunt WHERE id_abonne =
    (SELECT id_abonne FROM abonne WHERE prenom='chloe' ));
```

**Explications** : La dernière requête entre parenthèse s'exécute en premier et nous sort de la table abonné l'id d'abonné de Chloé (dans le cadre de cet entraînement, nous considérerons qu'il n'y a qu'une seule Chloé).  
`SELECT id_abonne FROM abonne WHERE prenom='chloe';`

Le résultat de cette requête est :

id_abonne
3

Ensuite, dans la requête du milieu (entre parenthèses) nous demandons à obtenir les id de livres ayant été empruntés par l'abonné n°3  
`SELECT id_livre FROM emprunt WHERE id_abonne = 3;`

Le résultat de cette requête est :

id_livre
100
105

Pour terminer, dans la requête du dessus, nous demandons à afficher les titres des livres qui NE SONT PAS les n° de livres récupérés soit le 100 et 105. En effet, nous ne voulons pas connaître les n° de livres que Chloé à emprunté mais tous les autres ! Cette requête permet d'abord de lister les n° de livres que chloé à emprunté pour ensuite les exclures (NOT IN) au moment de la demande des titres de livres.

```
SELECT titre FROM livre WHERE id_livre NOT IN(100,105);
```

Le résultat de cette requête est :

titre
Bel-Ami
Le père Goriot
Le Petit chose
La Reine Margot

Résultat  
: C'est donc le résultat final.

résultat

titre
Bel-Ami
Le père Goriot
Le Petit chose
La Reine Margot

Demande : Afficher le(s) titre de(s) livre(s) que Chloé n'a pas encore rendu(s) à la bibliothèque.

- Indices :
- Les titres des livres se trouvent dans la table livre.
  - Les prénoms des abonnés se trouvent dans la table abonné.
  - La liste des emprunts (qui a emprunté quoi ?) se trouve dans la table emprunt.
  - Nous ne pouvons pas relier la table abonné directement avec la table livre (car ces deux tables ne possèdent pas de champ en commun).
  - Nous pouvons relier la table livre avec la table emprunt. Nous pouvons aussi relier la table abonne avec la table emprunt.

Requête :

```
SQL - Requête Imbriquée - SELECT

SELECT titre FROM livre WHERE id_livre IN
  (SELECT id_livre FROM emprunt WHERE date_rendu IS NULL AND id_abonne =
    (SELECT id_abonne FROM abonne WHERE prenom='chloe' ));
```

Explications : La dernière requête entre parenthèse s'exécute en premier et nous sort de la table abonné l'id d'abonné de Chloé (dans le cadre de cet entraînement, nous considérerons qu'il n'y a qu'une seule Chloé).

```
SELECT id_abonne FROM abonne WHERE prenom='chloe';
```

Le résultat de cette requête est :

id_abonne
3

Ensuite, dans la requête du milieu (entre parenthèses) nous demandons à obtenir les id de livres ayant été empruntés par l'abonné n°3 et n'ayant pas été

rendus (date rendu is null)

```
SELECT id_livre FROM emprunt WHERE date_rendu IS NULL AND id_abonne = 3;
```

Le résultat de cette requête est :

id_livre
105

Pour terminer, dans la requête du dessus, nous demandons à afficher les titres des livres qui correspondent au n° (id) de livres récupérés (dans notre exemple, le 105)

```
SELECT titre FROM livre WHERE id_livre IN(105);
```

Dans le cadre de cette requête, même si un seul résultat aurait pu être compris par le signe égal (=), il était plus prudent de prévoir le mot clé IN dans la mesure où le nombre de résultats renvoyés par la requête du milieu peut changer à tout moment.

Le résultat de cette requête est :

titre
Les Trois Mousquetaires

#### Résultat

: C'est donc le résultat final.

résultat

titre
Les Trois Mousquetaires

**Demande** : Combien de livre Guillaume a emprunté à la bibliotheque ?

#### Indices :

- Nous ne souhaitons pas savoir quels sont les livres empruntés par Guillaume mais combien il en a emprunté.
- Nous devons prendre en compte la table abonné (pour connaître le n° d'abonné de Guillaume)
- Nous devons également prendre en compte la table emprunt puisque à l'intérieur tous les emprunts y sont repertoriés.
- Le champ en commun entre les deux tables est id\_abonne. Il servira donc de jonction dans le cadre de notre requête imbriquée pour assurer la correspondance des données.

#### Requête :

SQL - Requête Imbriquée - SELECT
<pre>SELECT COUNT(*) AS 'nombre de livre' FROM emprunt WHERE id_abonne = (SELECT id_abonne FROM abonne WHERE prenom='guillaume');</pre>

**Explications** : La dernière requête entre parenthèse s'exécute en premier et nous sort de la table abonné l'id d'abonné de Guillaume.

```
SELECT id_abonne FROM abonne WHERE prenom = 'guillaume';
```

Le résultat de cette requête est :

id_abonne
1

Ensuite, la première requête nous permet de compter (via le mot clé COUNT) tous les emprunts réalisés par cet abonné (le n°1).

```
SELECT COUNT(*) AS 'nombre de livre' FROM emprunt WHERE id_abonne = 1;
```

le mot clé AS permet de donner un alias pour renommer l'entête du résultat (cela reste facultatif mais toujours conseillé car plus lisible).

Le résultat de cette requête est :

nombre de livre
2

#### Résultat

: C'est donc le résultat final.

résultat

nombre de livre
2

## Jointure

49

### Les Jointures SQL

Tout comme les requêtes imbriquées, les jointures SQL permettent d'effectuer des requêtes sur plusieurs tables.

De la même manière que pour les requêtes imbriquées, si l'on souhaite pouvoir réaliser une jointure il faut qu'un champ commun se trouve dans les différentes tables concernées.

L'avantage des jointures est que l'on peut obtenir dans le résultat final des colonnes / champs issue de plusieurs tables différentes.

#### Décomposition d'une requête de jointure

Pour effectuer une jointure, nous respecterons le format suivant :

#### SQL - Requête de Jointure - SELECT

```
SELECT -- liste des champs que je souhaite afficher dans mon résultats
FROM -- de quelles table cela provient ? de quelle table vais-je avoir besoin ?
WHERE -- condition de jointure pour assurer le croisement des données.
AND/OR -- autre conditions éventuelles
```

Pour se mettre en jambe, nous allons prendre un cas concret sur notre base de données bibliotheque.

**Demande :** Afficher les dates auxquelles Guillaume s'est rendu à la bibliothèque pour emprunter ou pour rendre un livre.

**Indices :**

- Nous attendons un résultat à 3 colonnes avec les champs suivants : prenom - date\_sortie - date rendu

**Requête :**

#### SQL - Requête de Jointure - SELECT

```
SELECT a.prenom, e.date_sortie, e.date_rendu
FROM abonne a, emprunt e
WHERE a.id_abonne=e.id_abonne
```

```
AND a.prenom='guillaume';
```

Explications :

Ligne 1 - SELECT - Nous sélectionnons les champs que nous souhaitons obtenir dans le résultat, c'est à dire : prenom, date\_sortie, date\_rendu  
Sur cette même ligne, vous pourrez noter l'utilisation de préfixe a., e. Pour le moment ces préfixes ne correspondent à rien.

Ligne 2 - FROM - Nous annonçons les tables dont nous aurons besoin pour réussir notre requête, dans notre cas : abonne, emprunt.

Sur cette même ligne, vous pouvez constater la définition des préfixes "abonne a", "emprunt e", cela permet au système de comprendre que le "e" représente la table emprunt et que le "a" représente la table abonne.

L'avantage des préfixes est de pouvoir donner un nom plus court (1 ou plusieurs lettres) aux tables afin de les représenter, nous garderons à l'esprit que le format idéal c'est : bdd.table.champ.

Cela sera légèrement plus long à écrire mais cela aura l'avantage d'être clair pour quelqu'un qui nous rejoindrait sans connaître la modélisation par cœur (surtout dans le cas de modélisation complexe, ça aide).

Il faudra garder à l'esprit qu'on utilise d'abord les préfixes (sur la ligne n°1) avant de les définir (sur la ligne n°2)

Ligne 3 - WHERE - La condition WHERE permet d'assurer le croisement des données entre la table abonne et la table emprunt.

Nous passons par notre champ commun id\_abonne. C'est précisément ce qui permet d'effectuer notre jointure.

Ligne 4 - AND - La condition AND permet d'appliquer une condition complémentaire, dans notre cas nous souhaitons connaître les dates de passages de Guillaume.

Résultat

:

résultat

prenom	date_sortie	date_rendu
Guillaume	2014-12-17	2014-12-18
Guillaume	2014-12-19	2014-12-28

Si nous voulions arriver au même résultat avec une requête imbriquée cela ne serait pas possible puisque les colonnes / champs ne sont pas issues de la même table.

abonne	livre	livre
prenom	date_sortie	date_rendu
Guillaume	2014-12-17	2014-12-18
Guillaume	2014-12-19	2014-12-28

Pour qu'une requête imbriquée soit possible, il faut que les colonnes présentes dans le résultat final soient issues de la même table.

#### SQL - Requête Imbriquée - SELECT

```
SELECT date_sortie, date_rendu FROM emprunt WHERE id_abonne = (SELECT id_abonne FROM abonne WHERE prenom='guillaume');
```

Dans ce résultat, les colonnes du résultat sont issues de la même table (date\_sortie et date\_rendu proviennent d'une seule table : livre).

résultat

date_sortie	date_rendu
2014-12-17	2014-12-18
2014-12-19	2014-12-28



#### ➤ Informations

- nous pouvons tout faire avec une jointure, ce qui n'est pas le cas de la requête imbriquée.

- nous utiliserons quand même la requête imbriquée si nous visons uniquement des colonnes d'une seule table dans le résultat final (plus optimisé).

**Demande :** Afficher les dates auxquelles les livres écrits par Alphonse Daudet ont été empruntés ou rendus à la bibliothèque.

**Indices :**

- Nous attendons un résultat à 3 colonnes avec les champs suivants : date\_sortie - date rendu
- La jointure et la requête imbriquées sont possibles puisque les colonnes du résultat final sont proviennent de la même table.

**Requête :**

#### SQL - Requête de Jointure - SELECT

```
SELECT e.date_sortie, e.date_rendu
FROM livre l, emprunt e
WHERE l.id_livre=e.id_livre
AND l.auteur = 'Alphonse Daudet';
```

**Explications :**

Ligne 1 - SELECT - Nous sélectionnons les champs que nous souhaitons obtenir dans le résultat : date\_sortie, date\_rendu (nous utilisons des préfixes)

Ligne 2 - FROM - Nous annonçons les tables dont nous aurons besoin pour réussir notre requête, dans notre cas : livre, emprunt. (nous définissons les préfixes)

Ligne 3 - WHERE - La condition WHERE permet d'assurer le croisement des données entre la table livre et la table emprunt. Nous passons par notre champ commun id\_livre. C'est précisément ce qui permet d'effectuer notre jointure.

Ligne 4 - AND - La condition AND permet d'appliquer une condition complémentaire, dans notre cas nous souhaitons connaître les dates d'emprunt et dates de rendu de livre pour Alphonse Daudet.

**Résultat**

:

résultat

date_sortie	date_rendu
2014-12-19	2014-12-22

La même chose en requête imbriquée :

#### SQL - Requête Imbriquée - SELECT

```
SELECT date_sortie, date_rendu FROM emprunt WHERE id_livre IN ( SELECT id_livre FROM livre WHERE auteur='Alphonse Daudet');
```

**Demande :** Qui a emprunté le livre 'Une Vie' sur l'année 2014 ?

**Indices :**

- Nous attendons un résultat à 1 colonne avec le champ prenom
- La jointure et la requête imbriquées sont possibles puisque il n'y a qu'une seule colonne (donc forcément issue d'une seule table).

**Requête :**

#### SQL - Requête de Jointure - SELECT

```
SELECT a.prenom
FROM abonne a, emprunt e, livre l
WHERE l.id_livre = e.id_livre
AND e.id_abonne = a.id_abonne
AND l.titre = 'Une vie'
AND e.date_sortie LIKE '2014%';
```

Explications :

Ligne 1 - SELECT - Nous sélectionnons les champs que nous souhaitons obtenir dans le résultat : prenom (nous utilisons des préfixes)

Ligne 2 - FROM - Nous annonçons les tables dont nous aurons besoin pour réussir notre requête, dans notre cas : abonne, livre, emprunt. (nous définissons les préfixes)

Ligne 3 - WHERE - La condition WHERE permet d'assurer le croisement des données entre la table livre et la table emprunt. Nous passons par notre champ commun id\_livre.

Ligne 4 - AND - La condition AND permet également d'effectuer une jointure, cette fois-ci entre la table abonne et la table emprunt. Nous passons par notre champ commun id\_abonne.

Ligne 5 - AND - La condition AND permet de cibler le livre ayant pour titre "Une vie" (conformément à la demande de départ).

Ligne 6 - AND - La condition AND permet de cibler la date de sortie à l'année 2014 (conformément à la demande de départ).

Résultat

:

résultat

prenom
Guillaume
Chloe

La même chose en requête imbriquée :

#### SQL - Requête Imbriquée - SELECT

```
SELECT prenom FROM abonne WHERE id_abonne IN ( SELECT id_abonne FROM emprunt WHERE date_sortie LIKE '2014%' AND id_livre = (
SELECT id_livre FROM livre WHERE titre='Une vie'));
```

Demande : Afficher le nombre de livres empruntés par chaque abonné

Indices :

- Nous attendons un résultat à 2 colonnes avec le champ prenom et le nombre (le champ nombre n'existe pas, il faudra donc qu'il soit calculé)

Requête :

#### SQL - Requête de Jointure - SELECT

```
SELECT a.prenom, COUNT(e.id_livre) AS 'nombre de livre emprunte'
FROM abonne a, emprunt e
WHERE a.id_abonne=e.id_abonne
GROUP BY e.id_abonne;
```

Explications :

Ligne 1 - SELECT - Nous sélectionnons le champ prenom et comptons le nombre de références id\_livre dans la table emprunt (nous aurions pu choisir n'importe quel champ de la table emprunt pour ce calcul). Nous utilisons des préfixes.

Ligne 2 - FROM - Nous annonçons les tables dont nous aurons besoin pour réussir notre requête, dans notre cas : abonne, emprunt. (nous définissons les préfixes)

Ligne 3 - WHERE - La condition WHERE permet d'assurer le croisement des données entre la table abonne et la table emprunt. Nous passons par notre champ commun id\_abonne.

Ligne 4 - GROUP BY - Cette ligne nous permet de faire des regroupements pour répartir le calcul pour chacun des abonnés.

Résultat

:

résultat

--	--

prenom	nombre de livre emprunte
Guillaume	2
Benoit	3
Chloe	2
Laura	1

Demande : Qui a emprunté Quoi ? et Quand ?

Indices :

- Qui = prenom
- Quoi = titre de livre
- Quand = date\_sortie
- Nous attendrons un résultat avec 3 colonnes.

Requête :

SQL - Requête de Jointure - SELECT

```
SELECT a.prenom, l.titre, e.date_sortie
FROM abonne a, emprunt e, livre l
WHERE a.id_abonne=e.id_abonne
AND e.id_livre = l.id_livre;
```

Explications :

Ligne 1 - SELECT - Nous sélectionnons le champ prenom, titre, et date\_sortie.

Ligne 2 - FROM - Nous annonçons les tables dont nous aurons besoin pour réussir notre requête, dans notre cas : abonne, livre, emprunt.

Ligne 3 - WHERE - La condition WHERE permet d'assurer le croisement des données entre la table abonne et la table emprunt. Nous passons par notre champ commun id\_abonne.

Ligne 4 - AND - La condition AND permet également d'effectuer une jointure, cette fois-ci entre la table livre et la table emprunt. Nous passons par notre champ commun id\_livre.

Résultat

:

résultat

prenom	titre	date_sortie
Guillaume	Une vie	2014-12-17
Guillaume	La Reine Margot	2014-12-19
Benoit	Bel-Ami	2014-12-18
Benoit	Les Trois Mousquetaires	2015-03-20
Benoit	Une vie	2015-06-15
Chloe	Une vie	2014-12-19
Chloe	Les Trois Mousquetaires	2015-06-13
Laura	Le Petit chose	2014-12-19

Pas de requête imbriquée possible pour arriver à ce résultat.

Avant d'effectuer la prochaine requête, nous allons ajouter un abonné (qui par défaut, n'aura pas emprunté de livre) :



## SQL - Requête d'insertion - INSERT

```
INSERT INTO abonne (prenom) VALUES ('Alex');
SELECT * FROM abonne;
```

résultat

id_abonne	prenom
1	Guillaume
2	Benoit
3	Chloe
4	Laura
5	Alex

**Demande :** Afficher le prénom des abonnés avec le numéro des livres qu'ils ont emprunté

**Indices :**

- champs ciblés : prenom, id\_livre

**Requête :**

## SQL - Requête de Jointure - SELECT

```
SELECT ab.prenom, em.id_livre
FROM abonne ab, emprunt em
WHERE ab.id_abonne = em.id_abonne;
```

**Explications :**

Ligne 1 - SELECT - Nous sélectionnons le champ prenom et id\_livre.

Ligne 2 - FROM - Nous annonçons les tables dont nous aurons besoin pour réussir notre requête, dans notre cas : abonne, emprunt.

Ligne 3 - WHERE - La condition WHERE permet d'assurer le croisement des données entre la table abonne et la table emprunt. Nous passons par notre champ commun id\_abonne.

**Résultat**

:

résultat

prenom	id_livre
Guillaume	100
Guillaume	104
Benoit	101
Benoit	105
Benoit	100
Chloe	100
Chloe	105
Laura	103

Comme vous pouvez le constater, Alex notre dernier abonné ne se trouve pas parmi la liste de résultat.

Est-ce normal ? Oui c'est plutôt logique puisqu'il n'a emprunté aucun livre jusqu'à présent.

Il est quand même possible de retrouver sa trace via une jointure externe.

50

Jointure Externe

**Demande :** Afficher le prénom des abonnés avec le numéro des livres qu'ils ont emprunté

**Indices :**  
- champs ciblés : prenom, id\_livre

**Requête :**

SQL - Requête de Jointure (externe) - SELECT

```
SELECT abonne.prenom, emprunt.id_livre
FROM abonne LEFT JOIN emprunt
ON abonne.id_abonne = emprunt.id_abonne;
```

**Explications :**

Ligne 1 - SELECT - Nous selectionnons le champ prenom et id\_livre.

Ligne 2 - FROM - LEFT JOIN - La table placée à gauche de l'expression LEFT JOIN sera la table dont tous les résultats seront rapatriés (sans correspondance exigée dans l'autre table).  
Nous savons qu'Alex fera partie du résultat même s'il n'a pas emprunté de livre.

Ligne 3 - ON - La condition se fait par le mot clé ON et non plus WHERE dans le cadre d'une requête externe type LEFT JOIN, RIGHT JOIN (FULL JOIN n'étant pas valable sous Mysql).

**Résultat**  
:

résultat

prenom	id_livre
Guillaume	100
Guillaume	104
Benoit	101
Benoit	105
Benoit	100
Chloe	100
Chloe	105
Laura	103
Alex	NULL

La requête externe est donc plus complète car nous rapatrions tous les résultats de la table de gauche même s'il n'y a pas de correspondance dans l'autre

table dont nous nous servons pour effectuer la jointure.

Imaginez plus tard que dans votre site vous ayez des membres, des produits, et des commandes (les membres achètent des produits).  
Imaginez la situation suivante :

Table Membre	
id_membre	pseudo
28	Romain
29	Fred

Table Produit	
id_produit	titre
557	tshirt
558	pull

Table Commande			
id_commande	id_membre	id_produit	date
1802	28	557	2014
1803	29	558	2015

Si un membre supprime son compte (suppression dans la table membre), la commande restera présente dans la table commande avec l'inscription NULL :

Table Commande			
id_commande	id_membre	id_produit	date
1802	NULL	557	2014
1803	29	558	2015

Lorsque vous voudrez afficher le contenu de cette table dans un backOffice, vous pourrez alors avec la jointure externe demander à considérer comme table de gauche (avec LEFT JOIN) la table commande afin qu'aucune donnée ne soit oubliée (et plus particulièrement les commandes n'ayant pas de correspondance dans la table membre, même si le membre n'existe plus).

## Union

51

Union

Il est possible de fusionner 2 résultats en 1 seul grâce au mot clé UNION.

```
SQL - Requête de Jointure (externe) - SELECT
SELECT auteur AS 'liste personne physique' FROM livre UNION SELECT prenom FROM abonne ;
```

résultat

liste personne physique
GUY DE MAUPASSANT
HONORE DE BALZAC
ALPHONSE DAUDET
ALEXANDRE DUMAS

Guillaume
Benoit
Chloe
Laura
Alex

Explications : Dans cette requête, nous fusionnons la liste d'abonnés avec la liste des auteurs.  
Cela n'a pas d'intérêt dans l'exemple ci-dessus mais dans d'autres contextes cela peut être très pratique.

UNION se comporte comme un UNION DISTINCT par défaut. C'est à dire que les doublons seront éliminés.  
Si je souhaite obtenir toutes les valeurs (même celles qui sont en double), je devrais utiliser le mot clé UNION ALL.

## Fonction

52

Les fonctions

Une fonction est un morceau de code prévu pour s'exécuter et réaliser un traitement et/ou fournir un résultat.

Certaines fonctions sont prédéfinies (prévues et disponibles par le langage SQL) et d'autres peuvent être inscrites par le développeur (on appelle ça des *fonctions utilisateurs* ou en anglais : *user function*).

On ne doit pas connaître par cœur toutes les fonctions d'un langage (parce que ça ne sert à rien et parce que cela est mis à jour avec le temps) néanmoins un bon développeur est autonome et est capable de trouver son information tout seul via la doc technique [Mysql](#).

53

Fonctions prédéfinies

Voici quelques exemples de fonctions prédéfinies :

Fonction Prédéfinie	Description
<code>SELECT CURDATE();</code>	Affiche la date courante Pratique pour afficher des chambres d'hotel à réserver, ainsi personne n'aura à modifier la requête pour actualiser la date.
<code>SELECT CURDATE() + 0;</code>	Affiche la date courante sans tirets Ce format est pratique pour nommer un fichier à la date du jour.
<code>SELECT CURTIME();</code>	Affiche l'heure courante
<code>SELECT NOW();</code>	Affiche la date et l'heure courante
<code>SELECT DATE_ADD('2016-01-02', INTERVAL 31 DAY);</code>	Affiche une date future (avec 31 jours de plus).
<code>SELECT DATE_FORMAT('2012-10-03 22:23:00', '%d/%m/%Y - %H:%i:%s'); SELECT *, DATE_FORMAT(date_rendu, 'le %d/%m/%Y') FROM emprunt;</code>	Redéfinit le format de la date (format Français dans l'exemple)
<code>SELECT CONCAT('10-11-12', '12-11-10');</code>	

<code>SELECT CONCAT('a','b','c');</code>	Permet de concaténer (faire suivre) des informations
<code>SELECT CONCAT_WS(' - ','a','b','c');</code>	Permet de concaténer (faire suivre) des informations avec un séparateur WS = With Separator Cette fonction est très pratique pour réunir dans une même colonne plusieurs champs / colonnes différents comme l'adresse, le code postal et la ville.
<code>SELECT LENGTH('moi');</code>	Indique la longueur d'une chaîne (nombre de caractères)
<code>SELECT SUBSTRING('bonjour',4);</code>	Permet de couper une chaîne Dans notre exemple : à partir du 4e caractère.
<code>SELECT TRIM(' bonsoir ');</code>	Permet de supprimer les espaces en début et en fin de chaîne.
<code>SELECT DATABASE();</code>	Indique quelle base de données est en cours d'utilisation.
<code>SELECT LAST_INSERT_ID();</code>	Indique quel est le dernier identifiant généré par une base de données (après une requête INSERT).
<code>SELECT PASSWORD('mypass');</code>	Permet de crypter une chaîne sur l'algorithme AES. Cela peut être utilisé lors d'une insertion en table.

Bien entendu, il existe beaucoup d'autres fonctions SQL qui peuvent rendre service au développeur.

Une fonction utilisateur est créée par le développeur pour un besoin spécifique récurrent.

En effet, si une tâche ou un traitement précis est souvent effectué, il est souvent de bon augure d'isoler ces lignes de code dans une fonction que l'on pourra exécuter à tout moment plus facilement.

Voici le cycle à respecter pour se servir de fonctions utilisateur (fonctions personnalisées à nos besoins) :

- 1- La déclaration de la fonction (cela permet de la créer, si nous voulons l'utiliser plus tard il faut déjà qu'elle existe)
- 2- L'exécution de la fonction (cela correspond à la même démarche que pour les fonctions prédéfinies)

Imaginons que le comptable nous envoie toujours les informations des employés avec leur salaire brut mais que nous ayons pour consigne d'insérer le salaire net.

Ce calcul sera répétitif.

Voici une fonction qui consisterait à transformer le salaire brut d'un employé en salaire net :

*Dans cet exemple nous considérerons que le taux de différence entre le salaire brut et le salaire net est toujours de 20 %*

#### SQL / Fonction Utilisateur (personnalisé)

```
DELIMITER $
CREATE FUNCTION salaire_brut_en_net(sal INT) RETURNS INT
COMMENT 'Fonction permettant le calcul de salaire'
READS SQL DATA
BEGIN
    RETURN (sal*0.8);
END $
DELIMITER ;
```

Explications : Chaque ligne expliquée

```
DELIMITER $
```

Le delimiter permet de modifier le signe de reconnaissance de MYSQL du point-virgule (;) par le signe dollar (\$).

Nous modifions ce signe car nous devons mettre des points-virgules à certains moments de nos instructions (dans le corps de la fonction) et nous ne souhaitons pas que Mysql pense qu'il s'agit de la fin de notre fonction et qu'ils exécute un code qui ne serait pas terminé.

```
CREATE FUNCTION
```

permet de créer une fonction

```
salaire_brut_en_net
```

représente le nom de notre fonction (choisissez un nom explicite et en cohérence avec le traitement réalisé)

```
(sal INT)
```

représente un argument (paramètre) entrant de type INTeger (nombre)

Ceci signifie que notre traitement a besoin d'une information pour réaliser son traitement correctement.

Nous prévoyons l'argument entrant lors de la déclaration de la fonction mais celui-ci ne sera précisé qu'au moment de l'exécution de la fonction.

```
RETURNS INT
```

nous indiquons que notre fonction est destinée à retourner une valeur de type INTeger (nombre)

```
COMMENT 'Fonction permettant le calcul de salaire'
```

commentaire d'accompagnement pour mieux l'appréhender lors du listing des fonctions disponibles.

```
READS SQL DATA
```

Cette ligne permet d'indiquer au système que notre traitement ne fera que lire (et non pas modifier, supprimer) des données.

```
BEGIN
```

Début de nos instructions

```
RETURN (sal*0.8);
```

Nous retournerons le salaire multiplié par 0,8 %, c'est à dire que l'on retire 20 % et qu'on garde seulement 80 % du chiffre qu'on nous aura communiqué (avec l'argument entrant).

Return est aussi le mot clé permettant de renvoyer une valeur et généralement terminer une fonction

Dans notre cas, c'est à la fois le début et la fin de notre traitement (puisque'il s'agit de notre unique instruction).

Nous aurions pu mettre plusieurs lignes entre begin et end, cela est illimité.

```
END
```

Fin de nos instructions

```
DELIMITER ;
```

Permet de retrouver le delimiteur d'origine.

Pour tester si notre fonction fonctionne, il faut l'exécuter :

```
SQL / Fonction Utilisateur (personnalisé)
-- 1er exemple (test sur un nombre)
SELECT salaire_brut_en_net(2500);
-- 2e exemple (test lors d'une insertion)
INSERT INTO employes (prenom, salaire) values ('test', salaire_brut_en_net(3200));
SELECT * FROM employes;
-- 3e exemple (test lors d'une selection)
SELECT prenom, salaire_brut_en_net(salaire) FROM employes;
```

A travers ces exemples, nous constatons que l'on peut exécuter notre fonction dans différents contextes et plusieurs fois de suite.

Tant que cette fonction n'est pas supprimée, cette fonction restera en mémoire dans le système et sera utilisable sur la base de données sur laquelle elle a été créée.

Pour supprimer une fonction :

```
DROP FUNCTION salaire_brut_en_net ;
```

Pour consulter les fonctions déclarées / disponibles :

```
SHOW FUNCTION STATUS ;
```

## Table Virtuelle

55

Les tables virtuelles

Une table virtuelle se construit à partir d'une requête.

Généralement cela permet d'isoler des données résultant de jointure complexe et de relancer des opérations sur ces même résultats.

```
SQL / Table Virtuelle
CREATE VIEW vue_emprunt AS
SELECT a.prenom, l.titre, e.date_sortie
FROM abonne a, livre l, emprunt e
WHERE a.id_abonne = e.id_abonne
AND l.id_livre = e.id_livre ;
```

Cette jointure reste simple, parfois les requêtes font plus de 20 lignes avec des jonctions précises

Une table virtuelle s'appelle aussi VUE (d'où l'utilisation du code `CREATE VIEW`).

Nous pouvons constater sa présence dans la liste des tables :

```
SHOW TABLES ;
```

Nous pouvons consulter les données à l'intérieur

```
SELECT * FROM vue_emprunt ;
```

Consulter l'emplacement de sauvegarde réel des views :

```
SELECT * FROM information_schema.views ;
```

Supprimer une vue (table virtuelle) :

```
DROP view vue_emprunt ;
```



➤ Pour récapituler

- Nous sauvegardons uniquement la requête permettant de mener aux résultats.
- Une table virtuelle est une table qui se construit à partir d'une autre; Habituellement pour créer une table nous avons besoin de "CREATE TABLE...", une table virtuelle se construit à partir d'une requête et de colonnes de tables existantes.
- Une table virtuelle est pratique pour isoler des résultats suite à une jointure compliquée, cela permettra de faire des requêtes plus simples directement sur les résultats de la jointure.
- Une table virtuelle est constamment à jour et possède les mêmes données que la table d'origine puisqu'elle garde seulement la requête en mémoire. (Par conséquent, si je change un enregistrement dans la table virtuelle, ça change également dans la table d'origine, et vice-versa).

## Table Temporaire

56

### Les tables temporaires

Une table temporaire se construit à partir d'une requête et ré-enregistre des données existantes à un instant T.

Cela permet d'isoler une portion de données, il peut être utile de l'utiliser pour créer un sous-ensemble d'une autre table plus léger afin de soulager le serveur.

#### SQL / Table Temporaire

```
CREATE TEMPORARY TABLE emprunt2014 AS SELECT * FROM emprunt WHERE date_sortie LIKE '2014%' AND date_rendu LIKE '2014%' ;
```

*Cette requête nous permettra d'isoler les emprunts réalisés lors de l'année 2014.*

Nous ne la verrons pas dans la liste des tables :

```
SHOW TABLES ;
```

Nous pouvons consulter les données à l'intérieur

```
SELECT * FROM emprunt2014 ;
```

Nous n'avons pas réellement besoin de supprimer une table temporaire puisqu'elle est supprimée automatiquement à la fin d'une session.



➤ Pour récapituler

- Une table temporaire se construit à partir d'une requête et ré-enregistre des données existantes à un instant T.
- il peut être utile de l'utiliser pour créer un sous-ensemble d'une autre table plus léger afin de soulager le serveur.
- une table temporaire peut permettre de masquer certains aspects du schéma (confidentialité) ou d'en changer la complexité en ne montrant que certaines colonnes



- une table temporaire possède ses propres données (si je modifie la table temporaire, ça ne changera pas la table d'origine qui a "servi" à la construire, et vice-versa)
- une table temporaire est supprimée automatiquement lorsque l'on ferme la console (durée de vie liée à la session en cours)

## Transaction

57

### Les transactions

Lorsque nous utilisons des requêtes d'action qui impactent les données (type INSERT, UPDATE, DELETE), habituellement il n'est pas possible de faire marche arrière.

Faire une erreur peut arriver à tout le monde. Le moteur de stockage InnoDB propose une option d'annulation / confirmation des requêtes.

#### SQL / Transaction

```
START TRANSACTION;  
SELECT * FROM employes;  
UPDATE employes SET salaire = 123 WHERE id_employes = 699;  
SELECT * FROM employes;  
ROLLBACK; -- ou COMMIT;
```

`START TRANSACTION` permet de démarrer une zone de requête sous réserve de confirmation / annulation.

Si nous souhaitons annuler nos requêtes, il nous suffira d'inscrire : `ROLLBACK;`

Au contraire, si nous souhaitons confirmer nos requêtes, il nous faudra inscrire : `COMMIT;`

Que l'on choisisse `ROLLBACK` ou `COMMIT`, cela TERMINE la transaction, ça veut dire que les prochaines requêtes ne pourront pas être annulées, à moins de remettre un `START TRANSACTION`.



#### ➤ Attention

Si vous ne choisissez ni `ROLLBACK` ni `COMMIT`, ce sera `ROLLBACK` qui s'appliquera par défaut.

Pensez donc à valider si toutes vos requêtes sont bonnes.

Autre cas, si vous avez une transaction avec 5 requêtes : 2 bonnes, 1 mauvaise, 2 bonnes.

Vous voudrez certainement annuler la mauvaise requête mais le `ROLLBACK` va s'appliquer sur l'ensemble et annuler également les bonnes requêtes.

Pour cette raison il existe les transactions avancées que nous verrons au prochain chapitre.

58

### Transaction Avancée

Les transactions avancées permettent de créer des points de restauration.

#### SQL / Transaction

```
START TRANSACTION;  
SELECT * FROM employes;
```

```

SAVEPOINT point1;

UPDATE employes SET salaire = 1500 WHERE id_employes = 699;
SELECT * FROM employes;
SAVEPOINT point2;

UPDATE employes SET salaire = 2300 WHERE id_employes = 699;
SELECT * FROM employes;
SAVEPOINT point3;

UPDATE employes SET salaire = 1400 WHERE id_employes = 699;
SELECT * FROM employes;
SAVEPOINT point4;

ROLLBACK to point3; -- retour au point3
SELECT * FROM employes;
ROLLBACK to point4; -- ERREUR
ROLLBACK to point1; -- retour au point1

COMMIT; -- Valide et termine la transaction.

```

SAVEPOINT permet de créer un point de restauration sur lequel nous pourrions revenir plus tard.

S'il y a 10 points de restauration, je peux revenir au 6e puis ensuite au 3e

Je ne peux pas revenir au 3e et ensuite revenir au 6e (les points doivent être antérieur à l'instant où l'on se trouve).

## Variable et requête préparée

59

### Les Variables

Une variable est un espace nommé permettant de conserver une valeur.

Il existe 3 types de variables :

- Les variables systèmes (prévu par le langage)
- Les variables de portée globale (défini par le développeur)
- Les variables de portée locale (défini par le développeur dans le cadre de fonction ou procédure stockée)

Quelques exemples :

#### SQL / Variables

```

SHOW VARIABLES ; -- permet d'observer les variables système

SELECT @@version; -- les variables système comporte un double arrobasse @@

SET @ecole = "Mon Ecole"; -- Affectation d'une variable.
SELECT @ecole; -- Affichage du contenu de la variable. Les variables globales comportent un seul arrobasse @.

-- SET pays = "France"; SELECT pays; -- les variables locales (dans le cadre de fonction ou procédure) ne comportent pas d'arrobasse @.

```

60

### Les Requêtes préparées

Le cycle d'une requête classique est le suivant : Analyse / Interpretation / Exécution.

#### SQL / Requête Préparée

```

PREPARE req FROM 'SELECT * FROM employes WHERE service = "commercial";

```

Nous passons dans les étapes d'analyse et d'interprétation.

```
SQL / Requête Préparée
EXECUTE req;
EXECUTE req;
EXECUTE req;
EXECUTE req;
EXECUTE req;
```

Nous passons uniquement dans l'étape d'exécution.

Pour 5 exécutions d'une requête préparée, cela nous fait rentrer dans 7 étapes (1 analyse, 1 interprétation, 5 exécution).

Contrairement à une requête normale qui nous ferait 15 étapes (5 analyse, 5 interprétation, 5 exécution).

Nous pouvons également utiliser des requêtes préparées avec des arguments (paramètres variables) :

```
SQL / Requête Préparée
PREPARE req2 FROM 'SELECT * FROM abonne WHERE prenom = ?';
SET @prenom='laura';
EXECUTE req2 USING @prenom ;
```

Dans cet exemple, nous affectons la variable @prenom avec la valeur Laura. Nous l'utilisons ensuite dans le cadre de notre requête.

Pour supprimer une requête préparée :

```
SQL / Requête Préparée
DROP PREPARE req2;
```

## Procédure Stockée

61

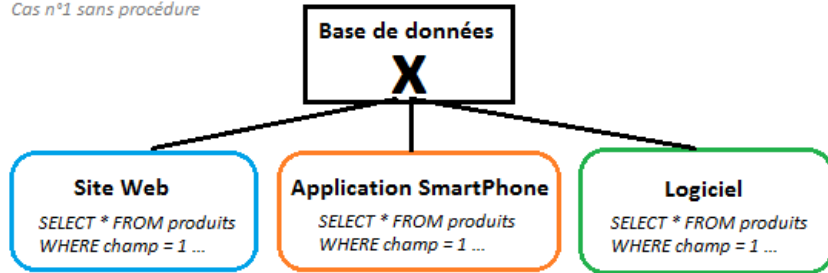
Procédure Stockée

Une procédure stockée est similaire à une fonction dans le sens où l'on peut isoler un traitement de plusieurs lignes de code à réaliser.

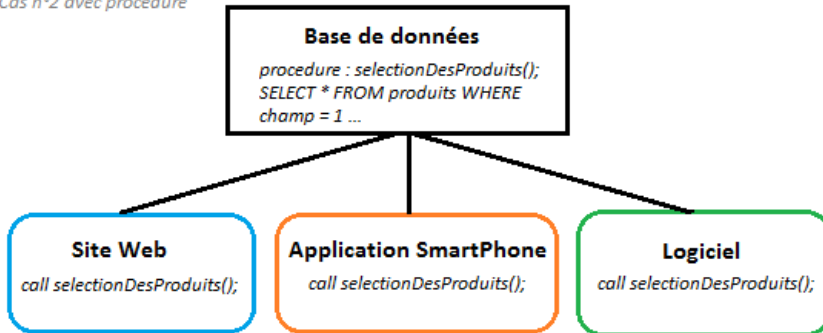
Dans une procédure stockée, nous ne sommes pas obligés de prévoir un RETURN contrairement à une fonction.

Dans le cadre d'un site web, application smartphone, logiciel, cela peut permettre de regrouper les requêtes au même endroit :

Cas n°1 sans procédure



Cas n°2 avec procédure



Dans le cas n°1, si l'une des requêtes est amené à changer, nous devons le modifier dans le site web (code php par exemple), dans l'application smartphone (code Objectiv-C par exemple), et dans le logiciel (code java par exemple).

Dans le cas n°2, la requête changera uniquement dans la procédure et cela laissera ce travail à l'administrateur / développeur de base de données.

Prenons un exemple pour avoir l'occasion d'écrire une procédure : Nous allons réunir les employés par groupe en fonction de leur salaire :

- Entre 3000 et + = Groupe 1
- Entre 2000 et 3000 = Groupe 2
- Entre 0 et 2000 = Groupe 3

#### SQL / Procédure Stockée

```
DELIMITER //
DROP PROCEDURE IF EXISTS groupe_employes //
DROP PROCEDURE groupe_employes //
CREATE PROCEDURE groupe_employes(IN pren VARCHAR(10))
BEGIN
    DECLARE s INT DEFAULT 0;
    SELECT * FROM employes WHERE prenom=pren;
    SELECT salaire FROM employes WHERE prenom = pren INTO s;
    IF s > 3000 THEN
        SELECT CONCAT(pren, ' fait partie du Groupe 1 avec ', s, ' de salaire mensuel') AS resultat;
    ELSEIF s >= 2000 AND s <= 3000 THEN
        SELECT CONCAT(pren, ' fait partie du Groupe 2 avec ', s, ' de salaire mensuel') AS resultat;
    ELSEIF s < 2000 AND s > 0 THEN
        SELECT CONCAT(pren, ' fait partie du Groupe 3 avec ', s, ' de salaire mensuel') AS resultat;
    ELSE
        SELECT CONCAT('employé inconnu', ' - ', s) AS resultat;
    END IF;
END //
CALL groupe_employes('Guillaume') //
CALL groupe_employes('Damien') //
CALL groupe_employes('Laura') //
CALL groupe_employes('mauvais prenom') //
```

Dans cet exemple, nous utiliserons des conditions IF, ELSEIF, ELSE pour définir le groupe d'un employé.

La procédure peut être rappelée à tout moment et avec nimportequel employé existant.

Pour supprimer une procédure :

```
DROP PROCEDURE [nom de la procedure] //
```

Pour observer les procédures existantes :

```
SHOW PROCEDURE STATUS//
SHOW PROCEDURE STATUS \G //
```

Un trigger (déclencheurs) permet d'isoler un morceau de code permettant de réaliser un traitement.

Contrairement aux fonctions et procédures stockées ce n'est pas nous qui l'exécuterons, il sera prévu pour s'exécuter seul, nous pourrions choisir l'action sur lequel le trigger s'exécutera.

Prenons un exemple pour avoir l'occasion d'écrire un trigger : Créons une table `employes_corbeille` qui sera la copie exacte en terme de structure (colonnes / champs) de la table `employés` (sans aucun enregistrement, à vide).

Nous allons faire en sorte d'enregistrer un employé dans `employes_corbeille` à chaque fois qu'il est supprimé de la table `employés` (sauf pour les commerciaux que nous ne garderons pas).

#### SQL / Trigger

```
DELIMITER |
DROP TRIGGER IF EXISTS t_employes_corbeille |
CREATE TRIGGER t_employes_corbeille AFTER DELETE ON employes
FOR EACH ROW
BEGIN
  IF OLD.service != 'commercial' then
    INSERT INTO employes_corbeille(id_employes, prenom, nom, sexe, service, date_embauche, salaire, id_secteur) VALUES (OLD.id_employes,
    OLD.prenom, OLD.nom, OLD.sexe, OLD.service, OLD.date_embauche, OLD.salaire, OLD.id_secteur);
  END IF;
END |
```

Comme vous pouvez le voir dans le cadre de ce trigger, nous avons précisé l'action `AFTER DELETE ON employes`

Ca veut dire qu'après chaque requête `DELETE` sur la table `employés`, ce trigger s'exécutera.

Nous réalisons une condition qui évite les commerciaux (employé du service commercial) pour insérer seulement tous les autres employés dans la table `employes_corbeille`.

Il ne vous reste plus qu'à faire le test en supprimant un employé de la table `employés`.

A la suite de cette action, vous devriez remarquer la présence de cet employé dans la table `employes_corbeille` (sauf s'il s'agit d'un commercial, il ne sera pas conservé).

## Evenement

Un event (événement) permet d'isoler un morceau de code permettant de réaliser un traitement à un moment précis.

Nous ne pourrions pas lancer son exécution mais nous pourrions choisir le moment dans le temps prévu pour commander son exécution.

Avant d'écrire un event, nous devons les activer :

#### SQL / Event

```
SHOW GLOBAL VARIABLES LIKE 'event_scheduler';
SET GLOBAL event_scheduler = 1 ;
SHOW GLOBAL VARIABLES LIKE 'event_scheduler';
```

Voici une procédure permettant de copier une table :

```

DELIMITER $$
DROP procedure IF EXISTS p_sauvegarde_employes $$
CREATE procedure p_sauvegarde_employes()
BEGIN
    SET @sql=concat('CREATE table copie_employes_',curdate()+0, ' SELECT * FROM employes');
    PREPARE req FROM @sql ;
    EXECUTE req ;
END $$

```

Voici maintenant un event capable d'exécuter cette procédure chaque jour :

```

DELIMITER $$
DROP event IF EXISTS e_sauvegarde_employes $$
CREATE EVENT e_sauvegarde_employes ON SCHEDULE EVERY 1 DAY STARTS '2016-12-01 15:10:00'
DO CALL p_sauvegarde_employes() $$

```

Nous prévoyons la planification de l'événement pour la première fois le 01/12/2016 et ensuite une fois par jour.

Cette ligne permet d'observer les événements :

```
SHOW EVENTS \G $$
```

## Evaluation (TP)

Fichier(s) utile(s) pour ce cours : [\\_immobilier.sql](#)

## Evaluation : Modélisation

Pour nous entrainer sur les requêtes, vous pouvez télécharger le fichier SQL complet ici : [\\_immobilier.sql](#)

Voici les structures et données enregistrées dans les tables :

Table Agence

Field	Type	Null	Key	Default	Extra
idAgence	int(6)	NO	PRI		auto_increment
nom	varchar(100)	NO			
adresse	varchar(100)	NO			

idAgence	nom	adresse
257400	logic-immo	www.logic-immo.com
383505	century21	rue century
504585	laforet	rue laforet
544688	fnaim	rue fnaim
608870	orpi	rue orpi
654178	fncia	rue fncia
654658	guy-hoquet	rue guy-hoquet
654893	seloger	www.seloger.com
692702	bouygues immobilier	www.bouygues-immobilier.net

## SQL / BDD Immobilier / Table Agence

```

CREATE TABLE IF NOT EXISTS agence (
  idAgence int(6) NOT NULL AUTO_INCREMENT,
  nom varchar(100) NOT NULL,
  adresse varchar(100) NOT NULL,
  PRIMARY KEY (idAgence)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;

INSERT INTO agence (idAgence, nom, adresse) VALUES
(257400, 'logic-immo', 'rue logic'),
(383505, 'century21', 'rue century'),
(504585, 'laforet', 'rue laforet'),
(544688, 'fnaim', 'rue fnaim'),
(608870, 'orpi', 'rue orpi'),
(654178, 'foncia', 'rue foncia'),
(654658, 'guy-hoquet', 'rue guy-hoquet'),
(654893, 'seloger', 'rue seloger'),
(692702, 'bouygues immobilier', 'rue bouygues');

```

Table Demande

Field	Type	Null	Key	Default	Extra
idDemande	int(5)	NO	PRI		auto_increment
idPersonne	int(3)	NO	MUL		
type	varchar(100)	NO			
ville	varchar(100)	NO			
budget	int(7)	NO			
superficie	int(5)	NO			
categorie	varchar(100)	NO			

idDemande	idPersonne	type	ville	budget	superficie	categorie
1	1	appartement	paris	530000	120	vente
2	3	appartement	bordeaux	120000	18	vente
3	4	appartement	bordeaux	145000	21	vente
4	5	appartement	bordeaux	152000	26	vente
5	6	appartement	lyon	200000	55	vente
6	9	appartement	paris	171000	40	vente
7	13	appartement	paris	163000	25	vente
8	16	appartement	paris	132000	15	vente
9	19	appartement	paris	350000	80	vente
10	22	appartement	lyon	600	20	location
11	25	appartement	lyon	188000	65	vente
12	27	appartement	paris	400	15	location
13	28	appartement	paris	330500	100	vente
14	31	appartement	paris	90000	15	vente
15	32	appartement	lyon	123800	21	vente
16	35	appartement	lyon	1200	70	vente

17	37	appartement	lyon	1500	100	vente
18	43	appartement	paris	600	20	location
19	44	appartement	paris	750	30	location
20	45	appartement	bordeaux	680	30	location
21	46	appartement	bordeaux	213000	40	vente
22	47	appartement	bordeaux	700	45	location
23	48	appartement	paris	195000	40	vente
24	49	appartement	paris	250000	60	vente
25	50	appartement	lyon	110000	12	vente
26	51	appartement	lyon	500	17	location
27	52	appartement	paris	800	40	location
28	53	appartement	paris	850	50	location
29	54	appartement	paris	177000	40	vente
30	55	appartement	paris	630	20	location

SQL / BDD Immobilier / Table Agence

```

CREATE TABLE IF NOT EXISTS demande (
  idDemande int(5) NOT NULL AUTO_INCREMENT,
  idPersonne int(3) NOT NULL,
  genre varchar(100) NOT NULL,
  ville varchar(100) NOT NULL,
  budget int(7) NOT NULL,
  superficie int(5) NOT NULL,
  categorie varchar(100) NOT NULL,
  PRIMARY KEY (idDemande),
  KEY idPersonne (idPersonne)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO demande (idDemande, idPersonne, genre, ville, budget, superficie, categorie) VALUES
(1, 1, 'appartement', 'paris', 530000, 120, 'vente'),
(2, 3, 'appartement', 'bordeaux', 120000, 18, 'vente'),
(3, 4, 'appartement', 'bordeaux', 145000, 21, 'vente'),
(4, 5, 'appartement', 'bordeaux', 152000, 26, 'vente'),
(5, 6, 'appartement', 'lyon', 200000, 55, 'vente'),
(6, 7, 'appartement', 'paris', 400000, 55, 'vente'),
(7, 9, 'appartement', 'paris', 171000, 40, 'vente'),
(8, 13, 'appartement', 'paris', 163000, 25, 'vente'),
(9, 16, 'appartement', 'paris', 132000, 15, 'vente'),
(10, 19, 'appartement', 'paris', 350000, 80, 'vente'),
(11, 22, 'appartement', 'lyon', 600, 20, 'location'),
(12, 25, 'appartement', 'lyon', 188000, 65, 'vente'),
(13, 27, 'appartement', 'paris', 400, 15, 'location'),
(14, 28, 'appartement', 'paris', 330500, 100, 'vente'),
(15, 31, 'appartement', 'paris', 90000, 15, 'vente'),
(16, 32, 'appartement', 'lyon', 123800, 21, 'vente'),
(17, 35, 'appartement', 'lyon', 1200, 70, 'vente'),
(18, 37, 'appartement', 'lyon', 1500, 100, 'vente'),
(19, 43, 'appartement', 'paris', 600, 20, 'location'),
(20, 44, 'appartement', 'paris', 750, 30, 'location'),
(21, 45, 'appartement', 'bordeaux', 680, 30, 'location'),
(22, 46, 'appartement', 'bordeaux', 213000, 40, 'vente'),
(23, 47, 'appartement', 'bordeaux', 700, 45, 'location'),
(24, 48, 'appartement', 'paris', 195000, 40, 'vente'),
(25, 49, 'appartement', 'paris', 250000, 60, 'vente'),
(26, 50, 'appartement', 'lyon', 110000, 12, 'vente'),
(27, 51, 'appartement', 'lyon', 500, 17, 'location'),
(28, 52, 'appartement', 'paris', 800, 40, 'location'),
(29, 53, 'appartement', 'paris', 850, 50, 'location'),
(30, 54, 'appartement', 'paris', 177000, 40, 'vente'),
(31, 55, 'appartement', 'paris', 630, 20, 'location');

```

Table Logement

Field	Type	Null	Key	Default	Extra
idLogement	int(4)	NO	PRI		auto_increment



type	varchar(100)	NO			
ville	varchar(100)	NO			
prix	int(7)	NO			
superficie	int(5)	NO			
categorie	varchar(100)	NO			

idLogement	type	ville	prix	superficie	categorie
5067	appartement	paris	185000	61	vente
5089	appartement	paris	115000	15	vente
5091	maison	paris	510000	130	vente
5122	appartement	bordeaux	550	17	location
5189	appartement	lyon	420	14	location
5245	appartement	paris	160000	40	vente
5246	appartement	paris	670	35	location
5249	appartement	lyon	110000	16	vente
5269	appartement	bordeaux	161500	33	vente
5278	appartement	paris	202000	90	vente
5324	appartement	lyon	690	31	location
5336	appartement	bordeaux	129600	27	vente
5378	appartement	bordeaux	121900	26	vente
5412	appartement	paris	680	40	location
5636	appartement	paris	150000	37	vente
5661	appartement	bordeaux	148600	36	vente
5723	appartement	bordeaux	170600	45	vente
5770	appartement	paris	139000	38	vente
5778	appartement	bordeaux	128600	43	vente
5779	appartement	paris	310000	105	vente
5786	appartement	paris	570	20	location
5860	appartement	bordeaux	105000	18	vente
5869	appartement	lyon	183600	60	vente
5873	appartement	lyon	176700	65	vente
5898	appartement	paris	690	40	location
5961	appartement	bordeaux	650	45	location
5963	appartement	paris	220000	60	vente

## SQL / BDD Immobilier / Table Logement

```
CREATE TABLE IF NOT EXISTS logement (
  idLogement int(4) NOT NULL AUTO_INCREMENT,
  genre varchar(100) NOT NULL,
  ville varchar(100) NOT NULL,
  prix int(7) NOT NULL,
  superficie int(5) NOT NULL,
  categorie varchar(100) NOT NULL,
  PRIMARY KEY (idLogement)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO logement (idLogement, genre, ville, prix, superficie, categorie) VALUES
(5067, 'appartement', 'paris', 685000, 61, 'vente'),
(5089, 'appartement', 'paris', 115000, 15, 'vente'),
(5091, 'maison', 'paris', 1510000, 130, 'vente'),
(5122, 'appartement', 'bordeaux', 550, 17, 'location'),
(5189, 'appartement', 'lyon', 420, 14, 'location'),
(5245, 'appartement', 'paris', 360000, 40, 'vente'),
(5246, 'appartement', 'paris', 970, 35, 'location'),
(5249, 'appartement', 'lyon', 110000, 16, 'vente'),
(5269, 'appartement', 'bordeaux', 171500, 33, 'vente'),
(5278, 'appartement', 'paris', 802000, 90, 'vente'),
(5324, 'appartement', 'lyon', 1090, 31, 'location'),
(5336, 'appartement', 'bordeaux', 229600, 27, 'vente'),
(5378, 'appartement', 'bordeaux', 121900, 26, 'vente'),
(5412, 'appartement', 'paris', 1680, 40, 'location'),
(5636, 'appartement', 'paris', 370000, 37, 'vente'),
(5661, 'appartement', 'bordeaux', 248600, 36, 'vente'),
(5723, 'maison', 'bordeaux', 370600, 45, 'vente'),
(5770, 'appartement', 'paris', 339000, 38, 'vente'),
(5778, 'appartement', 'bordeaux', 228600, 43, 'vente'),
(5779, 'appartement', 'paris', 1310000, 105, 'vente'),
(5786, 'appartement', 'paris', 570, 20, 'location'),
(5860, 'appartement', 'bordeaux', 98000, 18, 'vente'),
(5869, 'appartement', 'lyon', 683600, 60, 'vente'),
(5873, 'appartement', 'lyon', 676700, 65, 'vente'),
(5898, 'appartement', 'paris', 1890, 40, 'location'),
(5961, 'appartement', 'bordeaux', 2650, 45, 'location'),
(5963, 'appartement', 'paris', 520000, 60, 'vente'),
(5964, 'appartement', 'paris', 280000, 38, 'vente');
```

Table logement\_agence

Field	Type	Null	Key	Default	Extra
idLogementAgence	int(5)	NO	PRI		auto_increment
idAgence	int(6)	NO	MUL		
idLogement	int(4)	NO	MUL		
frais	int(7)	NO			

idLogementAgence	idAgence	idLogement	frais
1	257400	5067	15000
2	383505	5067	1000
3	257400	5089	8633
4	692702	5089	7623
5	654178	5091	28621
6	544688	5091	34564
7	654893	5122	700
8	608870	5189	350
9	257400	5245	10856
10	544688	5245	14230
11	608870	5246	800
12	257400	5249	16358
13	608870	5249	7625
14	257400	5269	9500
15	544688	5269	11890
16	544688	5278	25689
17	608870	5278	19653

18	544688	5324	600
19	544688	5336	9542
20	608870	5336	16985
21	504585	5378	8652
22	608870	5378	15230
23	257400	5412	680
24	544688	5636	5963
25	608870	5636	13654
26	654893	5661	9462
27	654178	5661	11656
28	608870	5723	16233
29	504585	5723	19654
30	692702	5770	13655
31	654178	5770	8903
32	383505	5778	6350
33	654658	5778	12655
34	654178	5779	26754
35	654658	5779	45032
36	654178	5786	898
37	383505	5786	520
38	257400	5860	12566
39	654658	5860	8905
40	544688	5869	23685
41	654893	5869	19321
42	257400	5873	13504
43	257400	5898	900
44	383505	5898	250
45	692702	5898	1300
46	257400	5961	1240
47	504585	5961	300
48	692702	5961	890
49	257400	5963	27542
50	692702	5963	42502
51	383505	5963	18455

## SQL / BDD Immobilier / Table logement\_agence

```
CREATE TABLE IF NOT EXISTS logement_agence (
  idLogementAgence int(5) NOT NULL AUTO_INCREMENT,
  idAgence int(6) NOT NULL,
  idLogement int(4) NOT NULL,
  frais int(7) NOT NULL,
  PRIMARY KEY (idLogementAgence),
  KEY idAgence (idAgence),
  KEY idLogement (idLogement)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;
```

```
INSERT INTO logement_agence (idLogementAgence, idAgence, idLogement, frais) VALUES
(1, 257400, 5067, 34250),
(2, 383505, 5067, 30000),
(3, 257400, 5089, 5750),
(4, 692702, 5089, 7623),
(5, 654178, 5091, 75500),
(6, 544688, 5091, 56050),
(7, 654893, 5122, 700),
(8, 608870, 5189, 350),
(9, 257400, 5245, 18856),
(10, 544688, 5245, 14230),
(11, 608870, 5246, 800),
(12, 257400, 5249, 5500),
(13, 608870, 5249, 7625),
(14, 257400, 5269, 9500),
(15, 544688, 5269, 8575),
(16, 544688, 5278, 25689),
(17, 608870, 5278, 40100),
(18, 544688, 5324, 600),
(19, 544688, 5336, 9542),
(20, 608870, 5336, 11480),
(21, 504585, 5378, 8652),
(22, 608870, 5378, 6095),
(23, 257400, 5412, 680),
(24, 544688, 5636, 18500),
(25, 608870, 5636, 13654),
(26, 654893, 5661, 9462),
(27, 654178, 5661, 11656),
(28, 608870, 5723, 16233),
(29, 504585, 5723, 19654),
(30, 692702, 5770, 13655),
(31, 654178, 5770, 16950),
(32, 383505, 5778, 11430),
(33, 654658, 5778, 12655),
(34, 654178, 5779, 65500),
(35, 654658, 5779, 45032),
(36, 654178, 5786, 898),
(37, 383505, 5786, 520),
(38, 257400, 5860, 4900),
(39, 654658, 5860, 8905),
(40, 544688, 5869, 23685),
(41, 654893, 5869, 34180),
(42, 257400, 5873, 33835),
(43, 257400, 5898, 900),
(44, 383505, 5898, 250),
(45, 692702, 5898, 1300),
(46, 257400, 5961, 1240),
(47, 504585, 5961, 300),
(48, 692702, 5961, 890),
(49, 257400, 5963, 27542),
(50, 692702, 5963, 26000),
(51, 383505, 5963, 18455);
```

Table logement\_personne

Field	Type	Null	Key	Default	Extra
idLogementPersonne	int(5)	NO	PRI		auto_increment
idPersonne	int(3)	NO	MUL		
idLogement	int(4)	NO	UNI		

idLogementPersonne	idPersonne	idLogement
1	40	5067
2	41	5089
3	42	5091
4	2	5122
5	39	5189
6	7	5245
7	8	5246
8	10	5249
9	18	5269

10	21	5278
11	17	5324
12	36	5336
13	20	5378
14	29	5412
15	24	5636
16	34	5661
17	14	5723
18	57	5770
19	26	5778
20	56	5779
21	12	5786
22	11	5860
23	23	5869
24	38	5873
25	33	5898
26	15	5961
27	30	5963

SQL / BDD Immobilier / Table logement\_personne

```
CREATE TABLE IF NOT EXISTS logement_personne (  
  idLogementPersonne int(5) NOT NULL AUTO_INCREMENT,  
  idPersonne int(3) NOT NULL,  
  idLogement int(4) NOT NULL,  
  PRIMARY KEY (idLogementPersonne),  
  UNIQUE KEY idLogement (idLogement),  
  KEY idPersonne (idPersonne)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;  
  
INSERT INTO logement_personne (idLogementPersonne, idPersonne, idLogement) VALUES  
(1, 40, 5067),  
(2, 41, 5089),  
(3, 42, 5091),  
(4, 2, 5122),  
(5, 39, 5189),  
(6, 7, 5245),  
(7, 8, 5246),  
(8, 10, 5249),  
(9, 18, 5269),  
(10, 21, 5278),  
(11, 17, 5324),  
(12, 36, 5336),  
(13, 20, 5378),  
(14, 29, 5412),  
(15, 24, 5636),  
(16, 34, 5661),  
(17, 14, 5723),  
(18, 57, 5770),  
(19, 26, 5778),  
(20, 56, 5779),  
(21, 12, 5786),  
(22, 11, 5860),  
(23, 23, 5869),  
(24, 38, 5873),  
(25, 33, 5898),  
(26, 15, 5961),  
(27, 30, 5963),  
(28, 56, 5964);
```

Table Personne

--	--	--	--	--	--	--

Field	Type	Null	Key	Default	Extra
idPersonne	int(3)	NO	PRI		auto_increment
prenom	varchar(100)	NO			

idPersonne	prenom
1	william
2	gaetan
3	mehdi
4	charles
5	brigitte
6	sarah
7	lucas
8	quentin
9	patrick
10	emmanuel
11	elodie
12	agathe
13	valentine
14	charlotte
15	alice
16	samuel
17	mathieu
18	noemie
19	simon
20	florian
21	clement
22	yvon
23	lea
24	chloe
25	camille
26	alexandre
27	julie
28	leo
29	antoine
30	lola
31	celia
32	anna
33	caroline
34	adele
35	sabrina
36	nathalie

37	franck
38	tom
39	johan
40	priscillia
41	assia
42	nathan
43	aurore
44	marie
45	oceane
46	enzo
47	ines
48	hugo
49	jonathan
50	axelle
51	morgane
52	melissa
53	kevin
54	ophelie
55	victoria
56	alexis
57	robin

## SQL / BDD Immobilier / Table Personne

```
CREATE TABLE IF NOT EXISTS personne (
  idPersonne int(3) NOT NULL AUTO_INCREMENT,
  prenom varchar(100) NOT NULL,
  PRIMARY KEY (idPersonne)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ;
```

```
INSERT INTO personne (idPersonne, prenom) VALUES
```

```
(1, 'william'),
(2, 'gaetan'),
(3, 'mehdi'),
(4, 'charles'),
(5, 'brigitte'),
(6, 'sarah'),
(7, 'lucas'),
(8, 'quentin'),
(9, 'patrick'),
(10, 'emmanuel'),
(11, 'elodie'),
(12, 'agathe'),
(13, 'valentine'),
(14, 'charlotte'),
(15, 'alice'),
(16, 'samuel'),
(17, 'mathieu'),
(18, 'noemie'),
(19, 'simon'),
(20, 'florian'),
(21, 'clement'),
(22, 'yvon'),
(23, 'lea'),
(24, 'chloe'),
(25, 'camille'),
(26, 'alexandre'),
(27, 'julie'),
(28, 'leo'),
(29, 'antoine'),
(30, 'lola'),
(31, 'celia'),
(32, 'anna'),
(33, 'caroline'),
(34, 'adele'),
(35, 'sabrina'),
(36, 'nathalie'),
```

```
(37, 'franck'),
(38, 'tom'),
(39, 'johan'),
(40, 'priscillia'),
(41, 'assia'),
(42, 'nathan'),
(43, 'auore'),
(44, 'marie'),
(45, 'oceane'),
(46, 'enzo'),
(47, 'ines'),
(48, 'hugo'),
(49, 'jonathan'),
(50, 'axelle'),
(51, 'morgane'),
(52, 'melissa'),
(53, 'kevin'),
(54, 'ophelie'),
(55, 'victoria'),
(56, 'alexis'),
(57, 'robin');
```

## Clés étrangères

### SQL / BDD Immobilier / Foreign Key

```
ALTER TABLE demande
ADD CONSTRAINT demande_ibfk_1 FOREIGN KEY (idPersonne) REFERENCES personne (idPersonne);

ALTER TABLE logement_agence
ADD CONSTRAINT logement_agence_ibfk_2 FOREIGN KEY (idLogement) REFERENCES logement (idLogement),
ADD CONSTRAINT logement_agence_ibfk_1 FOREIGN KEY (idAgence) REFERENCES agence (idAgence);

ALTER TABLE logement_personne
ADD CONSTRAINT logement_personne_ibfk_2 FOREIGN KEY (idPersonne) REFERENCES personne (idPersonne),
ADD CONSTRAINT logement_personne_ibfk_1 FOREIGN KEY (idLogement) REFERENCES logement (idLogement);
```

Fichier SQL complet ici : [immobilier.sql](#)

65

## Evaluation : Questions/Réponses - Les requêtes

Dans le cadre de cette évaluation, La question et la réponse (résultat) sont données, il vous suffit de trouver la requête qui mène de la question à la réponse.

Question 1 : Affichez le nom des agences

### SQL / BDD Immobilier / Requete n°1

```
SELECT nom FROM agence;
```

résultat 1 :

nom
logic-immo
century21
laforet
fnaim
orpi
fncia



guy-hoquet
seloger
bouygues immobilier

Question 2 : Affichez le numéro de l'agence « Orpi »

SQL / BDD Immobilier / Requete n°2

```
SELECT idAgence FROM agence WHERE nom='orpi';
```

résultat 2 :

idAgence
608870

Question 3 : Affichez le premier enregistrement de la table logement

SQL / BDD Immobilier / Requete n°3

```
SELECT * FROM logement LIMIT 0,1;
```

résultat 3 :

idLogement	genre	ville	prix	superficie	categorie
5067	appartement	paris	685000	61	vente

Question 4 : Affichez le nombre de logements (Alias : Nombre\_de\_logements)

SQL / BDD Immobilier / Requete n°4

```
SELECT COUNT(*) AS 'nombre de logements' FROM logement;
```

résultat 4 :

nombre de logements
28

Question 5 : Affichez les logements à vendre à moins de 150 000 € dans l'ordre croissant des prix:

SQL / BDD Immobilier / Requete n°5

```
SELECT * FROM logement WHERE prix < 150000 AND categorie = 'vente' ORDER BY prix;
```

résultat 5 :

idLogement	genre	ville	prix	superficie	categorie
5860	appartement	bordeaux	98000	18	vente

5249	appartement	lyon	110000	16	vente
5089	appartement	paris	115000	15	vente
5378	appartement	bordeaux	121900	26	vente

Question 6 : Affichez le nombre de logements à la location (alias : nombre)

SQL / BDD Immobilier / Requete n°6

```
SELECT COUNT(idLogement) as 'nombre' FROM logement WHERE categorie = 'location';
```

résultat 6 :

nombre
8

Question 7 : Affichez les villes différentes recherchées par les personnes demandeuses d'un logement

SQL / BDD Immobilier / Requete n°7

```
SELECT DISTINCT ville FROM demande;
```

résultat 7 :

ville
paris
bordeaux
lyon

Question 8 : Affichez le nombre de biens à vendre par ville

SQL / BDD Immobilier / Requete n°8

```
SELECT ville, COUNT(ville) as 'nombre' FROM demande WHERE categorie='vente' GROUP BY ville;
```

résultat 8 :

ville	nombre
bordeaux	4
lyon	5
paris	11

Question 9 : Quelles sont les id des logements destinés à la location ?

SQL / BDD Immobilier / Requete n°9

```
SELECT idLogement FROM logement WHERE categorie = 'location';
```

résultat 9 :

idLogement
5122
5189
5246
5324
5412
5786
5898
5961

Question 10 : Quels sont les id des logements entre 20 et 30m² ?

SQL / BDD Immobilier / Requete n°10

SELECT idLogement FROM logement WHERE superficie BETWEEN 20 AND 30;

résultat 10 :

idLogement
5336
5378
5786

Question 11 : Quel est le prix vendeur (hors commission) du logement le moins cher à vendre ? (Alias : prix minimum)

SQL / BDD Immobilier / Requete n°11

SELECT MIN(prix) AS 'prix minimum' FROM logement WHERE categorie = 'vente';

résultat 11 :

prix minimum
98000

Question 12 : Dans quelle ville se trouve les maisons à vendre ?

SQL / BDD Immobilier / Requete n°12

SELECT genre, ville FROM logement WHERE genre = 'maison';

résultat 12 :

genre	ville
maison	paris

maison	bordeaux
--------	----------

Question 13 : L'agence Orpi souhaite diminuer les frais qu'elle applique sur le logement ayant l'id « 5246 ». Passer les frais de ce logement de 800 à 730€

#### SQL / BDD Immobilier / Requete n°13

```
UPDATE logement_agence SET frais=730 WHERE idLogement = 5246 and idAgence = (SELECT idAgence FROM agence WHERE nom='orpi');
```

résultat 13 :

Query OK, 1 row affected

Question 14 : Quels sont les logements gérés par l'agence « laforet »

#### SQL / BDD Immobilier / Requete n°14

```
SELECT idLogement FROM logement_agence WHERE idAgence = (SELECT idAgence FROM agence WHERE nom = 'laforet');
```

résultat 14 :

idLogement
5378
5723
5961

Question 15 : Affichez le nombre de propriétaires dans la ville de Paris (Alias : Nombre)

#### SQL / BDD Immobilier / Requete n°15

```
SELECT COUNT(DISTINCT(lp.idPersonne)) AS 'nombre' FROM logement_personne lp, logement l WHERE lp.idLogement = l.idLogement AND l.ville = 'paris';
```

résultat 15 :

nombre
13

Question 16 : Affichez les informations des trois premières personnes souhaitant acheter un logement

#### SQL / BDD Immobilier / Requete n°16

```
SELECT p.*, d.* FROM personne p, demande d WHERE p.idPersonne = d.idPersonne AND d.categorie = 'vente' LIMIT 0,3;
```

résultat 16 :

idPersonne	prenom	idDemande	idPersonne	genre	ville	budget	superficie	categorie
1	william	1	1	appartement	paris	530000	120	vente
3	mehdi	2	3	appartement	bordeaux	120000	18	vente
4	charles	3	4	appartement	bordeaux	145000	21	vente

Question 17 : Affichez le prénom du vendeur pour le logement ayant la référence « 5770 »

SQL / BDD Immobilier / Requete n°17

```
SELECT p.prenom FROM personne p, logement_personne lp WHERE p.idPersonne = lp.idPersonne and lp.idLogement = 5770;
```

résultat 17 :

prenom
robin

Question 18 : Affichez les prénoms des personnes souhaitant accéder à un logement sur la ville de Lyon

SQL / BDD Immobilier / Requete n°18

```
SELECT p.prenom FROM personne p, demande d where p.idPersonne = d.idPersonne AND d.ville = 'lyon';
```

résultat 18 :

prenom
sarah
yvon
camille
anna
sabrina
franck
axelle
morgane

Question 19 : Affichez les prénoms des personnes souhaitant accéder à un logement en location sur la ville de Paris

SQL / BDD Immobilier / Requete n°19

```
SELECT p.prenom FROM personne p, demande d where p.idPersonne = d.idPersonne AND d.ville = 'paris' AND d.categorie = 'location';
```

résultat 19 :

prenom
julie
aurore
marie
melissa
kevin
victoria

Question 20 : Affichez les prénoms des personnes souhaitant acheter un logement de la plus grande à la plus petite superficie

SQL / BDD Immobilier / Requete n°20

```
SELECT p.prenom, d.superficie FROM personne p, demande d WHERE p.idPersonne = d.idPersonne AND d.categorie='vente' ORDER BY d.superficie DESC;
```

résultat 20 :

prenom	superficie
william	120
leo	100
simon	80
sabrina	70
camille	65
jonathan	60
lucas	55
sarah	55
hugo	40
enzo	40
ophelie	40
patrick	40
brigitte	26
valentine	25
charles	21
anna	21
mehdi	18
samuel	15
celia	15
axelle	12

Question 21 : Quel sont les prix finaux proposés par les agences pour la maison à la vente ayant la référence « 5091 » ? (Alias : prix frais d'agence inclus)

SQL / BDD Immobilier / Requete n°21

```
SELECT (l.prix+la.frais) AS 'prix frais d'agence inclu' FROM logement l, logement_agence la WHERE l.idLogement = la.idLogement and la.idLogement = 5091;
```

résultat 21 :

prix frais d'agence inclus
1585500
1566050

Question 22 : Indiquez les frais ajoutés par l'agence immobilière pour le logement ayant la référence « 5873 » ?

SQL / BDD Immobilier / Requete n°22

```
SELECT l.idLogement, l.prix, la.frais, (l.prix+la.frais) as 'prix total' FROM logement l, logement_agence la WHERE la.idLogement = 5873 AND la.idLogement=l.idLogement;
```

résultat 22 :

idLogement	prix	frais	prix total
5873	676700	33835	710535

Question 23 : Si l'ensemble des logements étaient vendus ou loués demain, quel serait le bénéfice généré grâce aux frais d'agence et pour chaque agence (Alias : benefice, classement : par ordre croissant des gains)

SQL / BDD Immobilier / Requete n°23

```
SELECT a.nom, SUM(la.frais) AS benefice FROM agence a, logement_agence la WHERE a.idAgence = la.idAgence GROUP BY la.idAgence ORDER BY benefice;
```

résultat 23 :

nom	benefice
laforet	28606
seloger	44342
bouygues immobilier	49468
century21	60655
guy-hoquet	66592
orpi	96337
logic-immo	142953
fnaim	156871
foncia	170504

Question 24 : Affichez les id des biens en location, les prix, suivis des frais d'agence (classement : dans l'ordre croissant des prix) :

SQL / BDD Immobilier / Requete n°24

```
SELECT a.nom, l.idLogement, la.frais FROM logement_agence la, logement l, agence a WHERE la.idLogement = l.idLogement AND l.categorie='location' AND la.idAgence=a.idAgence ORDER BY l.prix;
```

résultat 24 :

nom	idLogement	frais
orpi	5189	350
seloger	5122	700
foncia	5786	898
century21	5786	520
orpi	5246	800

fnaim	5324	600
logic-immo	5412	680
century21	5898	250
bouygues immobilier	5898	1300
logic-immo	5898	900
logic-immo	5961	1240
laforet	5961	300
bouygues immobilier	5961	890

Question 25 : Quel est le prénom du propriétaire proposant le logement le moins cher à louer ?

SQL / BDD Immobilier / Requete n°25

SELECT prenom FROM personne WHERE idPersonne IN (SELECT idPersonne FROM logement\_personne WHERE idLogement IN (SELECT idLogement FROM logement where categorie='location' AND prix = (SELECT MIN(prix) FROM logement WHERE categorie='location')));

résultat 25 :

prenom
johan

Question 26 : Affichez le prénom et la ville où se trouve le logement de chaque propriétaire

SQL / BDD Immobilier / Requete n°26

SELECT p.prenom, l.ville FROM personne p, logement l, logement\_personne lp WHERE p.idPersonne = lp.idPersonne AND l.idLogement = lp.idLogement;

résultat 26 :

prenom	ville
priscillia	paris
assia	paris
nathan	paris
gaetan	bordeaux
johan	lyon
lucas	paris
quentin	paris
emmanuel	lyon
noemie	bordeaux
clement	paris
mathieu	lyon
nathalie	bordeaux
florian	bordeaux
antoine	paris
chloe	paris



adele	bordeaux
charlotte	bordeaux
robin	paris
alexandre	bordeaux
alexis	paris
agathe	paris
elodie	bordeaux
lea	lyon
tom	lyon
caroline	paris
alice	bordeaux
lola	paris
alexis	paris

Question 27 : Quel est l'agence immobilière s'occupant de la plus grande gestion de logements répertoriés à Paris ? (alias : nombre, classement : trié par ordre décroissant)

#### SQL / BDD Immobilier / Requete n°27

```
SELECT a.nom, COUNT(l.ville) as 'nombre' FROM agence a, logement_agence la, logement l WHERE la.idAgence = a.idAgence and la.idLogement=l.idLogement and l.ville = 'paris' GROUP BY a.nom ORDER BY nombre DESC;
```

résultat 27 :

nom	nombre
logic-immo	6
fncia	4
fnaim	4
century21	4
bouygues immobilier	4
orpi	3
guy-hoquet	1

Question 28 : Affichez le prix et le prénom des vendeurs dont les logements sont proposés à 130000 € ou moins en prix final avec frais appliqués par les agences (alias : prix final, classement : ordre croissant des prix finaux) :

#### SQL / BDD Immobilier / Requete n°28

```
SELECT p.prenom, (l.prix+la.frais) as 'prix final'
FROM personne p, logement l, logement_agence la, logement_personne lp
WHERE (l.prix+la.frais) <= 130000
AND p.idPersonne=lp.idPersonne
AND l.idLogement=la.idLogement
AND l.categorie='vente'
AND lp.idLogement=l.idLogement
AND lp.idLogement=la.idLogement
ORDER BY (l.prix+la.frais);
```

résultat 28 :

--	--

prenom	prix final
elodie	102900
elodie	106905
emmanuel	115500
emmanuel	117625
assia	120750
assia	122623
florian	127995

Question 29 : Affichez le nombre de logements à la vente dans la ville de recherche de « hugo » (alias : nombre)

SQL / BDD Immobilier / Requete n°29

```
SELECT COUNT(l.idLogement) as 'nombre'
FROM personne p, demande d, logement l
WHERE p.idPersonne = d.idPersonne
AND d.ville = l.ville
AND p.prenom='hugo'
AND l.categorie='vente';
```

résultat 29 :

nombre
10

Question 30 : Affichez le nombre de logements à la vente dans la ville de recherche de « hugo » et dans la superficie minimum qu’il attend ou dans une superficie supérieure (alias : nombre):

SQL / BDD Immobilier / Requete n°30

```
SELECT COUNT(l.idLogement) as 'nombre'
FROM personne p, demande d, logement l
WHERE p.idPersonne = d.idPersonne
AND d.ville = l.ville
AND d.superficie <= l.superficie
AND p.prenom='hugo'
AND l.categorie='vente';
```

résultat 30 :

nombre
6

Question 31 : Affichez le nombre d’opportunités d’achats dans la ville de recherche de « hugo » dans la superficie minimum qu’il attend ou dans une superficie supérieure et en prenant en compte tous ses autres critères de sélection (alias : nombre):

SQL / BDD Immobilier / Requete n°31

```
SELECT COUNT(l.idLogement) as 'nombre'
FROM personne p, demande d, logement l, logement_agence la, agence a
WHERE d.genre = l.genre
AND d.ville = l.ville
AND d.budget >= (l.prix+la.frais)
AND d.superficie <= l.superficie
AND la.idLogement = l.idLogement
AND p.idPersonne = d.idPersonne
AND l.categorie='vente'
AND a.idAgence = la.idAgence
AND p.prenom='hugo';
```

réultat 31 :

nombre
2

Question 32 : Affichez les prénoms des personnes souhaitant accéder à un logement en location sur la ville de Paris

SQL / BDD Immobilier / Requete n°32

```
SELECT p.prenom, d.genre AS 'genre recherche', d.ville AS 'ville recherche', d.budget AS 'budget max', d.superficie AS 'superficie min', d.categorie AS 'categorie recherche', l.idLogement, a.nom AS 'agence', l.genre AS 'genre propose', l.ville AS 'ville propose', (l.prix+la.frais) AS 'prix final', l.superficie AS 'superficie propose', l.categorie AS 'categorie propose'
FROM demande d, logement l, logement_agence la, personne p, agence a
WHERE d.genre = l.genre
AND d.ville = l.ville
AND d.budget >= (l.prix+la.frais)
AND d.superficie <= l.superficie
AND la.idLogement = l.idLogement
AND p.idPersonne = d.idPersonne
AND l.categorie='vente'
AND a.idAgence = la.idAgence
AND p.prenom='hugo';
```

réultat 32 :

pre nom	genre recherche	ville recherche	budget max	superficie min	categorie recherche	idLogement	agence	genre propose	ville propose	prix final	superficie propose	categorie propose
hugo	appartement	paris	495000	40	vente	5245	logic-immo	appartement	paris	378856	40	vente
hugo	appartement	paris	495000	40	vente	5245	fnaim	appartement	paris	374230	40	vente

Question 33 : En prenant en compte le « fichier client » avec leurs critères de sélection répertoriés sur la table « demande », quelle est l’agence immobilière susceptible de faire le plus de ventes ? (alias : nombre)

SQL / BDD Immobilier / Requete n°33

```
SELECT a.nom as 'agence', COUNT(a.nom) as 'nombre'
FROM demande d, logement l, logement_agence la, personne p, agence a
WHERE d.genre = l.genre
AND d.ville = l.ville
AND d.budget >= (l.prix+la.frais)
AND d.superficie <= l.superficie
AND la.idLogement = l.idLogement
AND p.idPersonne = d.idPersonne
AND l.categorie='vente'
AND a.idAgence = la.idAgence
GROUP BY a.nom
ORDER BY nombre DESC;
```

réultat 33 :

agence	nombre
logic-immo	6
bouygues immobilier	4
century21	3
orpi	2
guy-hoquet	2
fnaim	2

laforet	2
---------	---

**Question 34** : Affichez les prénoms des personnes cherchant un logement ainsi que les noms des agences (s'occupant de la gestion des logements) pour une mise en relation dans le cadre d'une susceptible location immobilière (tout en affichant les informations qui permettront de mettre en évidence une première année d'éventuels contrats, voir résultat).

#### SQL / BDD Immobilier / Requete n°34

```
SELECT p.prenom, d.genre as 'genre recherche', d.ville as 'ville recherche', (d.budget*12) as 'budget premiere annee', d.superficie as 'superficie min', d.categorie as 'categorie recherche', a.nom as 'agence', l.idLogement, l.genre as 'genre propose', l.ville as 'ville propose', (l.prix*12)+la.frais as 'prix premiere annee', l.superficie as 'superficie propose', l.categorie as 'categorie propose'
FROM demande d, logement l, logement_agence la, personne p, agence a
WHERE d.genre = l.genre
AND d.ville = l.ville
AND (d.budget*12) >= (l.prix*12)+la.frais
AND d.superficie <= l.superficie
AND la.idLogement = l.idLogement
AND p.idPersonne = d.idPersonne
AND l.categorie='location'
AND d.categorie='location'
AND a.idAgence = la.idAgence;
```

résultat 34 :

pre nom	genre recherche	ville recherche	budget premiere annee	superficie min	categorie recherche	agence	idLogement	genre propose	ville propose	prix premiere annee	superficie propose	categorie propose
victoria	appartement	paris	7560	20	location	century21	5786	appartement	paris	7360	20	location

**Question 35** : Affichez les prénoms des acheteurs potentiels, les prénoms des vendeurs ainsi que les agences s'occupant de la gestion de leurs logements pour une mise en relation dans le cadre d'une susceptible vente immobilière (tout en affichant les informations qui permettront de mettre en évidence cette éventuelle transaction, voir résultat).

#### SQL / BDD Immobilier / Requete n°35

```
SELECT p.prenom as 'acheteur', d.genre as 'genre recherche', d.ville as 'ville recherche', d.budget as 'budget max', d.superficie as 'superficie min', d.categorie as 'categorie recherche', a.nom as 'agence', p2.prenom as 'vendeur', l.genre as 'genre propose', l.ville as 'ville propose', (l.prix+la.frais) as 'prix final', l.superficie as 'superficie propose', l.categorie as 'categorie propose'
FROM demande d, logement l, logement_agence la, personne p, personne p2, agence a, logement_personne lp
WHERE d.genre = l.genre
AND l.idLogement = lp.idLogement
AND lp.idPersonne = p2.idPersonne
AND d.ville = l.ville
AND d.budget >= (l.prix+la.frais)
AND d.superficie <= l.superficie
AND la.idLogement = l.idLogement
AND p.idPersonne = d.idPersonne
AND l.categorie=d.categorie
AND a.idAgence = la.idAgence
AND a.idAgence = la.idAgence ;
```

résultat 35 :

acheteur	genre recherche	ville recherche	budget max	superficie min	categorie recherche	agence	vendeur	genre propose	ville propose	prix final	superficie propose	categorie propose
mehdi	appartement	bordeaux	120000	18	vente	logic-immo	elodie	appartement	bordeaux	102900	18	vente
mehdi	appartement	bordeaux	120000	18	vente	guy-hoquet	elodie	appartement	bordeaux	106905	18	vente
charles	appartement	bordeaux	145000	21	vente	laforet	florian	appartement	bordeaux	130552	26	vente
charles	appartement	bordeaux	145000	21	vente	orpi	florian	appartement	bordeaux	127995	26	vente
brigitte	appartement	bordeaux	172000	26	vente	laforet	florian	appartement	bordeaux	130552	26	vente
brigitte	appartement	bordeaux	172000	26	vente	orpi	florian	appartement	bordeaux	127995	26	vente
lucas	appartement	paris	600000	55	vente	logic-immo	lola	appartement	paris	547542	60	vente

lucas	appartement	paris	600000	55	vente	century21	lola	appartement	paris	538455	60	vente
lucas	appartement	paris	600000	55	vente	bouygues immobilier	lola	appartement	paris	546000	60	vente
samuel	appartement	paris	162000	15	vente	logic-immo	assia	appartement	paris	120750	15	vente
samuel	appartement	paris	162000	15	vente	bouygues immobilier	assia	appartement	paris	122623	15	vente
celia	appartement	paris	145000	15	vente	logic-immo	assia	appartement	paris	120750	15	vente
celia	appartement	paris	145000	15	vente	bouygues immobilier	assia	appartement	paris	122623	15	vente
enzo	appartement	bordeaux	413000	40	vente	century21	alexandre	appartement	bordeaux	240030	43	vente
enzo	appartement	bordeaux	413000	40	vente	guy-hoquet	alexandre	appartement	bordeaux	241255	43	vente
hugo	appartement	paris	495000	40	vente	logic-immo	lucas	appartement	paris	378856	40	vente
hugo	appartement	paris	495000	40	vente	fnaim	lucas	appartement	paris	374230	40	vente
jonathan	appartement	paris	650000	60	vente	logic-immo	lola	appartement	paris	547542	60	vente
jonathan	appartement	paris	650000	60	vente	century21	lola	appartement	paris	538455	60	vente
jonathan	appartement	paris	650000	60	vente	bouygues immobilier	lola	appartement	paris	546000	60	vente
ophelie	appartement	paris	377500	40	vente	fnaim	lucas	appartement	paris	374230	40	vente

Question 36 : Supprimer la personne n°idPersonne 13 (Valentine).

SQL / BDD Immobilier / Requete n°36

DELETE FROM personne WHERE idPersonne = 13 ;

résultat 36 :

Query OK

Question 37 : Afficher toutes les demandes enregistrées avec la personne à l'origine de la demande (Afficher également les demandes d'anciennes personnes n'existant plus dans notre base de données).

SQL / BDD Immobilier / Requete n°37

SELECT p.prenom, d.superficie, d.genre, d.ville, d.budget, d.categorie FROM personne p LEFT JOIN demande d ON p.idPersonne = d.idPersonne ;

résultat 37 :

prenom	superficie	genre	ville	budget	categorie
william	120	appartement	paris	530000	vente
gaetan					
mehdi	18	appartement	bordeaux	120000	vente
charles	21	appartement	bordeaux	145000	vente
brigitte	26	appartement	bordeaux	172000	vente
sarah	55	appartement	lyon	450000	vente
lucas	55	appartement	paris	600000	vente
quentin					

patrick	40	appartement	paris	371000	vente
emmanuel					
elodie					
agathe					
valentine	25	appartement	paris	253000	vente
charlotte					
alice					
samuel	15	appartement	paris	162000	vente
mathieu					
noemie					
simon	80	appartement	paris	720000	vente
florian					
clement					
yvon	20	appartement	lyon	680	location
lea					
chloe					
camille	65	appartement	lyon	558000	vente
alexandre					
julie	15	appartement	paris	490	location
leo	100	appartement	paris	1100000	vente
antoine					
lola					
celia	15	appartement	paris	145000	vente
anna	21	appartement	lyon	123800	vente
caroline					
adele					
sabrina	70	appartement	lyon	690000	vente
nathalie					
franck	100	appartement	lyon	1500	location
tom					
johan					
priscillia					
assia					
nathan					
aurore	20	appartement	paris	600	location
marie	30	appartement	paris	750	location
oceanne	30	appartement	bordeaux	680	location
enzo	40	appartement	bordeaux	413000	vente
ines	45	appartement	bordeaux	700	location
hugo	40	appartement	paris	495000	vente
jonathan	60	appartement	paris	650000	vente

axelle	12	appartement	lyon	110000	vente
morgane	17	appartement	lyon	500	location
melissa	40	appartement	paris	800	location
kevin	50	appartement	paris	850	location
ophelie	40	appartement	paris	377500	vente
victoria	20	appartement	paris	630	location
alexis					
robin					

Question 38 : Afficher toutes les personnes enregistrées avec leur demandes correspondantes (Afficher également les personnes n'ayant pas formulé de demandes).

SQL / BDD Immobilier / Requete n°36

```
SELECT p.prenom, d.superficie, d.genre, d.ville, d.budget, d.categorie FROM demande d LEFT JOIN personne p ON p.idPersonne = d.idPersonne ;
```

résultat 38 :

prenom	superficie	genre	ville	budget	categorie
william	120	appartement	paris	530000	vente
mehdi	18	appartement	bordeaux	120000	vente
charles	21	appartement	bordeaux	145000	vente
brigitte	26	appartement	bordeaux	172000	vente
sarah	55	appartement	lyon	450000	vente
lucas	55	appartement	paris	600000	vente
patrick	40	appartement	paris	371000	vente
	25	appartement	paris	253000	vente
samuel	15	appartement	paris	162000	vente
simon	80	appartement	paris	720000	vente
yvon	20	appartement	lyon	680	location
camille	65	appartement	lyon	558000	vente
julie	15	appartement	paris	490	location
leo	100	appartement	paris	1100000	vente
celia	15	appartement	paris	145000	vente
anna	21	appartement	lyon	123800	vente
sabrina	70	appartement	lyon	690000	vente
franck	100	appartement	lyon	1500	location
aurore	20	appartement	paris	600	location
marie	30	appartement	paris	750	location
oceane	30	appartement	bordeaux	680	location
enzo	40	appartement	bordeaux	413000	vente
ines	45	appartement	bordeaux	700	location
hugo	40	appartement	paris	495000	vente

jonathan	60	appartement	paris	650000	vente
axelle	12	appartement	lyon	110000	vente
morgane	17	appartement	lyon	500	location
melissa	40	appartement	paris	800	location
kevin	50	appartement	paris	850	location
ophelie	40	appartement	paris	377500	vente
victoria	20	appartement	paris	630	location

Question 39 : Affichez toutes les personnes enregistrées avec leur demandes correspondantes (Afficher également les personnes n'ayant pas formulé de demandes ainsi que les demandes d'anciennes personnes n'existant plus dans notre base de données).

SQL / BDD Immobilier / Requete n°39

SELECT p.prenom, d.superficie, d.genre, d.ville, d.budget, d.categorie FROM demande d LEFT JOIN personne p ON p.idPersonne = d.idPersonne
UNION DISTINCT
SELECT p.prenom, d.superficie, d.genre, d.ville, d.budget, d.categorie FROM personne p LEFT JOIN demande d ON p.idPersonne = d.idPersonne ;

résultat 39 :

prenom	superficie	genre	ville	budget	categorie
william	120	appartement	paris	530000	vente
mehdi	18	appartement	bordeaux	120000	vente
charles	21	appartement	bordeaux	145000	vente
brigitte	26	appartement	bordeaux	172000	vente
sarah	55	appartement	lyon	450000	vente
lucas	55	appartement	paris	600000	vente
patrick	40	appartement	paris	371000	vente
	25	appartement	paris	253000	vente
samuel	15	appartement	paris	162000	vente
simon	80	appartement	paris	720000	vente
yvon	20	appartement	lyon	680	location
camille	65	appartement	lyon	558000	vente
julie	15	appartement	paris	490	location
leo	100	appartement	paris	1100000	vente
celia	15	appartement	paris	145000	vente
anna	21	appartement	lyon	123800	vente
sabrina	70	appartement	lyon	690000	vente
franck	100	appartement	lyon	1500	location
aurore	20	appartement	paris	600	location
marie	30	appartement	paris	750	location
oceane	30	appartement	bordeaux	680	location
enzo	40	appartement	bordeaux	413000	vente
ines	45	appartement	bordeaux	700	location
hugo	40	appartement	paris	495000	vente



jonathan	60	appartement	paris	650000	vente
axelle	12	appartement	lyon	110000	vente
morgane	17	appartement	lyon	500	location
melissa	40	appartement	paris	800	location
kevin	50	appartement	paris	850	location
ophelie	40	appartement	paris	377500	vente
victoria	20	appartement	paris	630	location
gaetan					
quentin					
emmanuel					
elodie					
agathe					
charlotte					
alice					
mathieu					
noemie					
florian					
clement					
lea					
chloe					
alexandre					
antoine					
lola					
caroline					
adele					
nathalie					
tom					
johan					
priscillia					
assia					
nathan					
alexis					
robin					

Question 40 : Afficher la liste des personnes ayant plusieurs logements à vendre.

SQL / BDD Immobilier / Requete n°40

?

```

SELECT p.prenom
FROM personne p, logement l, logement_personne lp
WHERE p.idPersonne = lp.idPersonne
AND lp.idLogement = l.idLogement
GROUP BY lp.idPersonne
HAVING COUNT(lp.idPersonne) > 1 ;

```

résultat 40 :

prenom
alexis

Question 41 : Afficher la liste des personnes avec le logement qu'elles vendent, ou la demande de logement qu'elles recherchent.

#### SQL / BDD Immobilier / Requete n°41

```
SELECT p.idPersonne, p.prenom, d.genre, d.ville, d.budget, d.superficie, d.categorie, l.genre, l.ville, l.prix, l.superficie, l.categorie
FROM personne p LEFT JOIN logement_personne lp ON p.idPersonne = lp.idPersonne
LEFT JOIN demande d ON p.idPersonne = d.idPersonne
LEFT JOIN logement l ON l.idLogement = lp.idLogement ;
```

résultat 41 :

idPersonne	prenom	genre demandé	ville demandé	budget demandé	superficie demandé	categorie demandé	genre proposé	ville proposé	prix proposé	superficie proposé	categorie proposé
1	william	appartement	paris	530000	120	vente					
2	gaetan						appartement	bordeaux	550	17	location
3	mehdi	appartement	bordeaux	120000	18	vente					
4	charles	appartement	bordeaux	145000	21	vente					
5	brigitte	appartement	bordeaux	172000	26	vente					
6	sarah	appartement	lyon	450000	55	vente					
7	lucas	appartement	paris	600000	55	vente	appartement	paris	360000	40	vente
8	quentin						appartement	paris	970	35	location
9	patrick	appartement	paris	371000	40	vente					
10	emmanuel						appartement	lyon	110000	16	vente
11	elodie						appartement	bordeaux	98000	18	vente
12	agathe						appartement	paris	570	20	location
13	valentine	appartement	paris	253000	25	vente					
14	charlotte						maison	bordeaux	370600	45	vente
15	alice						appartement	bordeaux	2650	45	location
16	samuel	appartement	paris	162000	15	vente					
17	mathieu						appartement	lyon	1090	31	location
18	noemie						appartement	bordeaux	171500	33	vente
19	simon	appartement	paris	720000	80	vente					
20	florian						appartement	bordeaux	121900	26	vente
21	clement						appartement	paris	802000	90	vente
22	yvon	appartement	lyon	680	20	location					
23	lea						appartement	lyon	683600	60	vente
24	chloe						appartement	paris	370000	37	vente
25	camille	appartement	lyon	558000	65	vente					
26	alexandre						appartement	bordeaux	228600	43	vente
27	julie	appartement	paris	490	15	location					
28	leo	appartement	paris	1100000	100	vente					

29	antoine						appartement	paris	1680	40	location
30	lola						appartement	paris	520000	60	vente
31	celia	appartement	paris	145000	15	vente					
32	anna	appartement	lyon	123800	21	vente					
33	caroline						appartement	paris	1890	40	location
34	adele						appartement	bordeaux	248600	36	vente
35	sabrina	appartement	lyon	690000	70	vente					
36	nathalie						appartement	bordeaux	229600	27	vente
37	franck	appartement	lyon	1500	100	location					
38	tom						appartement	lyon	676700	65	vente
39	johan						appartement	lyon	420	14	location
40	priscillia						appartement	paris	685000	61	vente
41	assia						appartement	paris	115000	15	vente
42	nathan						maison	paris	1510000	130	vente
43	aurore	appartement	paris	600	20	location					
44	marie	appartement	paris	750	30	location					
45	oceane	appartement	bordeaux	680	30	location					
46	enzo	appartement	bordeaux	413000	40	vente					
47	ines	appartement	bordeaux	700	45	location					
48	hugo	appartement	paris	495000	40	vente					
49	jonathan	appartement	paris	650000	60	vente					
50	axelle	appartement	lyon	110000	12	vente					
51	morgane	appartement	lyon	500	17	location					
52	melissa	appartement	paris	800	40	location					
53	kevin	appartement	paris	850	50	location					
54	ophelie	appartement	paris	377500	40	vente					
55	victoria	appartement	paris	630	20	location					
56	alexis						appartement	paris	1310000	105	vente
56	alexis						appartement	paris	280000	38	vente
57	robin						appartement	paris	339000	38	vente

Question 42 : Afficher la liste des personnes ayant à la fois un logement à vendre ou louer et une demande d'achat ou de location.

#### SQL / BDD Immobilier / Requete n°42

```
SELECT p.prenom
FROM personne p, logement l, logement_personne lp, demande d
WHERE p.idPersonne = lp.idPersonne
AND lp.idLogement = l.idLogement
AND d.idPersonne = p.idPersonne
GROUP BY lp.idPersonne
HAVING COUNT(lp.idPersonne) > 0 AND COUNT(d.idPersonne) > 0;
```

résultat 42 :

prenom
--------

## Contrainte d'intégrité

---

CONTENU MANQUANT...

© Copyright Aucune reproduction, même partielle (textes, documents, images, etc.), ne peut être faite sans l'accord de son auteur.

**Liam TARDIEU**

**EVOGUE.fr - EPROJET.fr**