

Semifir

MySQL

semifir.com
03 20 52 20 68
13 Avenue du Président John F. Kennedy,
59000 Lille.
contact@semifir.com

Chapitres

Sommaire

1. Introduction aux bases de données
2. Notions fondamentales
3. Création de tables via MySQL Workbench
4. Clefs primaires et étrangères
5. Les jointures
6. Le temps et les fonctions
7. Les sous-requêtes



1. Introduction aux bases de données



Introduction aux bases de données

- Définition d'une Base de données :
 - entité dans laquelle il est possible de stocker des données de façon structurée plus ou moins reliées entre elles et avec le moins de redondances possible.
- A quoi sert une base de données :
 - Stocker des informations sous forme de tableau,
 - Lier des informations entre elles,
 - Manipuler et accéder aux informations.

Basiquement : Excel sous stéroïdes

Introduction aux bases de données

- Le **SGDB** (**Système de Gestion de Base de Données**) :
 - Est un programme qui permet la manipulation d'une base de données de manière efficace et collaborative.
- Quelques exemples de SGBD connus :
 - MySQL (Orienté Relationnel)
 - MariaDB
 - PosgreSQL
 - Oracle Database
 - Etc...
- Chaque SGBD possède ses propres avantages :
 - Coût,
 - Performances,
 - Versatilité
 - Langage et fonctions (dévié du SQL)

Introduction aux bases de données

- On va pouvoir y insérer :
 - Des tableaux en deux dimensions (une ligne entête, des lignes de data)
 - Des « clefs » d'identification uniques
 - Pas de doublons possibles,
 - Garantit que chaque objet soit référencé de manière unique
 - Des contraintes de typage
 - Imposer l'usage d'un nombre, d'une date dans une colonne
 - Imposer la longueur maximale d'une données présente dans une colonne
 - Des relations entre tables
 - Permettant de référencer un objet présent dans une autre table par son ID

Globalement : Un tableau croisé dynamique sous Excel <3

Introduction aux bases de données

Les moteurs :

- Ensemble d'algorithmes permettant de stocker et d'accéder aux données
- Un SGBD a souvent un moteur unique, MySQL laisse le choix
- Chaque moteur a ses avantages et inconvénients
 - Transactionnel ?
 - SGBD imposé ?

Name	Vendor	License	Transactional	Under active development	MySQL versions	MariaDB versions
Archive	Oracle	GPL	No	Yes	5.0 - present	5.1 - present
Aria	MariaDB	GPL	No	Yes	None	5.1 - present
Berkeley DB	Oracle	AGPLv3	Yes	No	? - 5.0	None
BLACKHOLE	Oracle	GPL	No	Yes	5.0 - present	5.1 - present
CONNECT	MariaDB	GPL	No	Yes	None	10.0 - present
CSV	Oracle	GPL	No	Yes	5.0 - present	5.1 - present
Falcon	Oracle	GPL	Yes	No	?	None
Federated	Oracle	GPL	?	No	5.0 - present	?
FederatedX	MariaDB	GPL	Yes	No	None	? - present
ColumnStore (formerly InfiniDB)	Calpont	GPL	Yes	Yes	None	10.5.4 - present
InnoDB	Oracle	GPL	Yes	Yes	3.23 - present	5.1 - present
MEMORY	Oracle	GPL	No	Yes	3.23 - present	5.1 - present
Mroonga	Groonga Project	GPL	No	Yes	None	10.0 - present
MyISAM	Oracle	GPL	No	No	3.23 - present	5.1 - present
MyRocks	Facebook	GPLv2	Yes	Yes	None	10.2 - present
NDB	Oracle	GPLv2	Yes	Yes	?	None
OQGRAPH	Oracle	GPLv2	No	No	None	5.2 - present
S3	MariaDB	GPL	No	Yes	None	10.5 - present
SEQUENCE	MariaDB	GPL	No	Yes	None	10.0 - present
Sphinx	Sphinx Technologies Inc.	GPL	No	No	None	5.2 - present
SPIDER	Kentoku Shiba	GPL	Yes	Yes	None	10.0 - present
TempTable	Oracle	GPL	No	Yes	8.0 - present	None
TokuDB	Percona	Modified GPL	Yes	No	None	5.5 - present
XtraDB	Percona	GPL	Yes	Yes	None	5.1 - 10.1

Introduction aux bases de données

- Exemple de table simple à deux dimensions :

ID	Nom	Prénom	Date de naissance	Classe
1	Bowie	David	08/01/1947	Musique
2	Elton	John	25/03/1947	Musique
3	Cartman	Éric	01/07/1989	CE2
4	Norton	Edward	18/08/1969	Théâtre
5	Nicks	Stevie	26/05/1948	Chant
6	Kilmister	Lemmy	24/12/1945	Basse

Introduction aux bases de données

- Exemple de table simple à deux dimensions, avec jointure :

Table élèves :

ID	Nom	Prénom	Date de naissance	Classe_ID
1	Bowie	David	08/01/1947	1 (<i>Musique</i>)
2	Elton	John	25/03/1947	1 (<i>Musique</i>)
3	Cartman	Éric	01/07/1989	5 (<i>CE2</i>)
4	Norton	Edward	18/08/1969	4 (<i>Théâtre</i>)
5	Nicks	Stevie	26/05/1948	2 (<i>Chant</i>)
6	Kilmister	Lemmy	24/12/1945	3 (<i>Basse</i>)

Table classes	
ID	Classe
1	Musique
2	Chant
3	Basse
4	Théâtre
5	CE2

Les tables ‘élèves’ et ‘classes’ ont été reliées ensemble par leur colonne ‘Classe_ID’ et ‘Classe’. On remplace alors les intitulés de leur classe par des numéros qu’on ira traduire d’une table à l’autre par une ‘jointure’.

2. Notions fondamentales



Notions fondamentales

Les types d'objets :

- Servent à imposer un type d'objet pour chacune de nos colonnes, ainsi :
 - Une colonne typée en INTEGER (entier) ne pourra contenir que des nombres entiers,
 - Une colonne typée en DATE ne pourra contenir que des dates,
 - Etc.
- On retrouve les mêmes types primaires que dans la plupart des langages de programmation
- Sur le principe :
 - Chaque ligne de notre table serait en quelque sorte un objet,
 - Chaque colonne serait un paramètre de cet objet

Semifir

Notions fondamentales

Les nombres entiers

Type de donnée	Spéc	Nombre d'octets
TINYINT	- 128 à 127	1
SMALLINT	-3278 à 3277	2
MEDIUMINT	-8388608 à 8388607	3
INT	-2147483648 à 2147483647	4
BIGINT	-9223372036854775808 à + 9223372036854775807	8

On peut préciser l'argument UNSIGNED, qui indique que notre nombre sera toujours positif. Le minimum passe à zéro et le maximum 'double' en conséquence. (ex : TINYINT UNSIGNED = 0 à 255)

Notions fondamentales

Les nombres flottants (avec des virgules)

Type de donnée	Spéc
FLOAT	Nombre flottant, précis sur 23 chiffres
DOUBLE	Nombre flottant de 24 à 53 chiffres
DECIMAL	Type « double », mais stocké en tant que STRING

***DECIMAL peut contenir jusqu'à 5 chiffres significatifs :
2 avant la virgule et 3 après***

Semifir

Notions fondamentales

Les chaines de caractères

Type de donnée	Description	Nombre d'octets
CHAR(x)	Le nombre entre () définira l'espace qu'il prendra	=x
VARCHAR(x)	Prendra 1 octet par lettre +1, dans la limite de (x)	$1L+1 \leq x$
TINYTEXT	Caractères limités par le nombre d'octet	2^8
TEXT	Caractères limités par le nombre d'octet	2^{16}
MEDIUMTEXT	Caractères limités par le nombre d'octet	2^{24}
LONGTEXT	Caractères limités par le nombre d'octet	2^{32}

$2^{32} = 4 \text{ Milliards. A titre de comparaison, le livre « Les Misérables » fait env. } 500K \text{ de mots (soit env. 3 millions de caractères)}$

Notions fondamentales

Exemple :

Comparaison entre CHAR(x) et VARCHAR(x)

Avantage : la taille de notre base de données est plus stable, mais plus volumineuse

Input :	CHAR(10)	Octets	VARCHAR(10)	Octets
« »	« »	10	« »	1
« Lic »	« Lic »	10	« Lic »	4
« Licorne »	« Licorne »	10	« Licorne »	8
« Vive les licornes »	« Vive les l»	10	« Vive les l»	11

Avantage : La taille de notre base est moins volumineuse, mais plus variable

La valeur entre parenthèses sert à indiquer le nombre maximum d'octets.

Grossièrement : 1 caractères = 1 Octet (et 1 Octet = 8 Bits).

Les caractères accentués prennent 2 octets

Notions fondamentales

Les types d'objets 'DATE'

Type de donnée	Spéc
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD Hh:mm:ss
TIME	Hh:mm:ss
YEAR	YYYY
TIMESTAMP	YYYYMMDDHhmmss

*Il existe aussi le UNIX_TIMESTAMP qui commence le 1970-01-01 00:00:00
Il représente combien de temps s'est écoulé depuis cette date.*

Notions fondamentales

Les conventions SQL

- **Nom de tables et colonnes :**

- Pas d'espaces,
 - Remplacés par des underscores
- Pas de caractères spéciaux
- Pas de 'mots réservés' (ex DATE, TEXT, TYPE, TABLE etc.)
 - Techniquement possible dans un STRING, mais très mauvaise pratique
- Utiliser des noms représentatifs du contenu, sans abréviations
- Privilégier le singulier

- **Syntaxe des commandes :**

- Mots clefs en TOUTE MAJUSCULE
- Noms des bases, tables et colonnes en minuscule
- Les options facultatives des commandes entre []

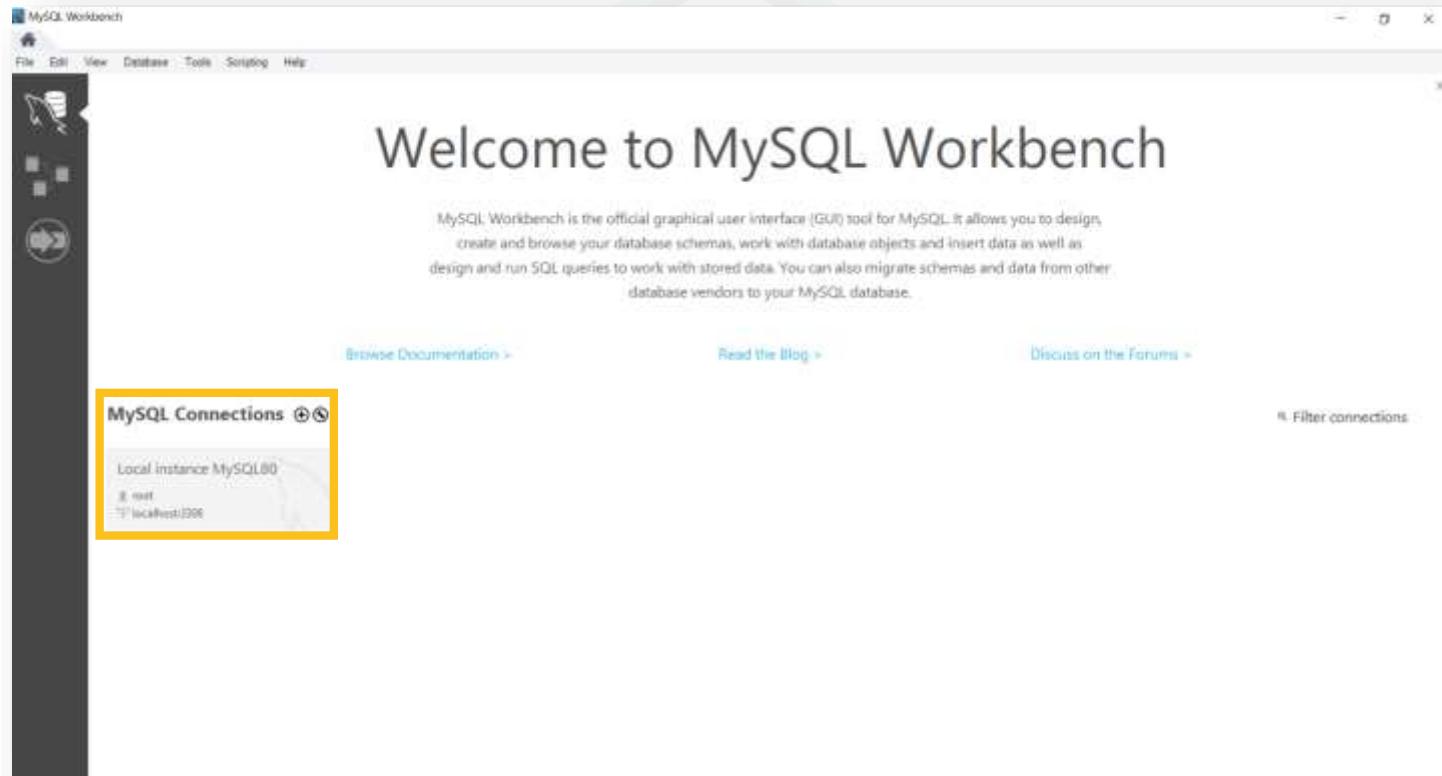
Plus d'informations sur : <https://sql.sh/1396-nom-table-colonne>

3. Création d'une base de données sur MySQL Workbench



Création d'une base de données sur MySQL Workbench

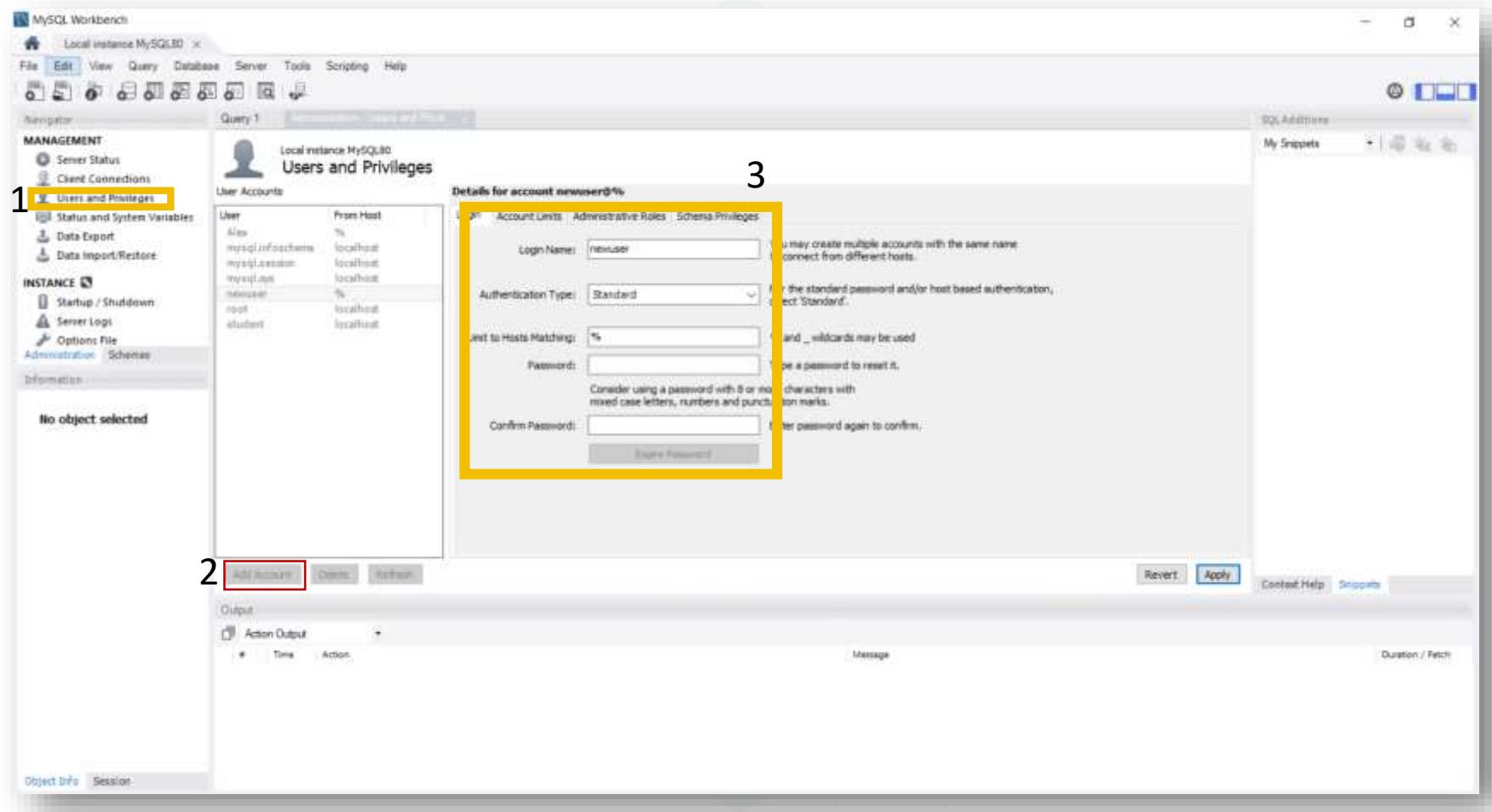
Sélection de la BDD



Semifir

Création d'une base de données sur MySQL Workbench

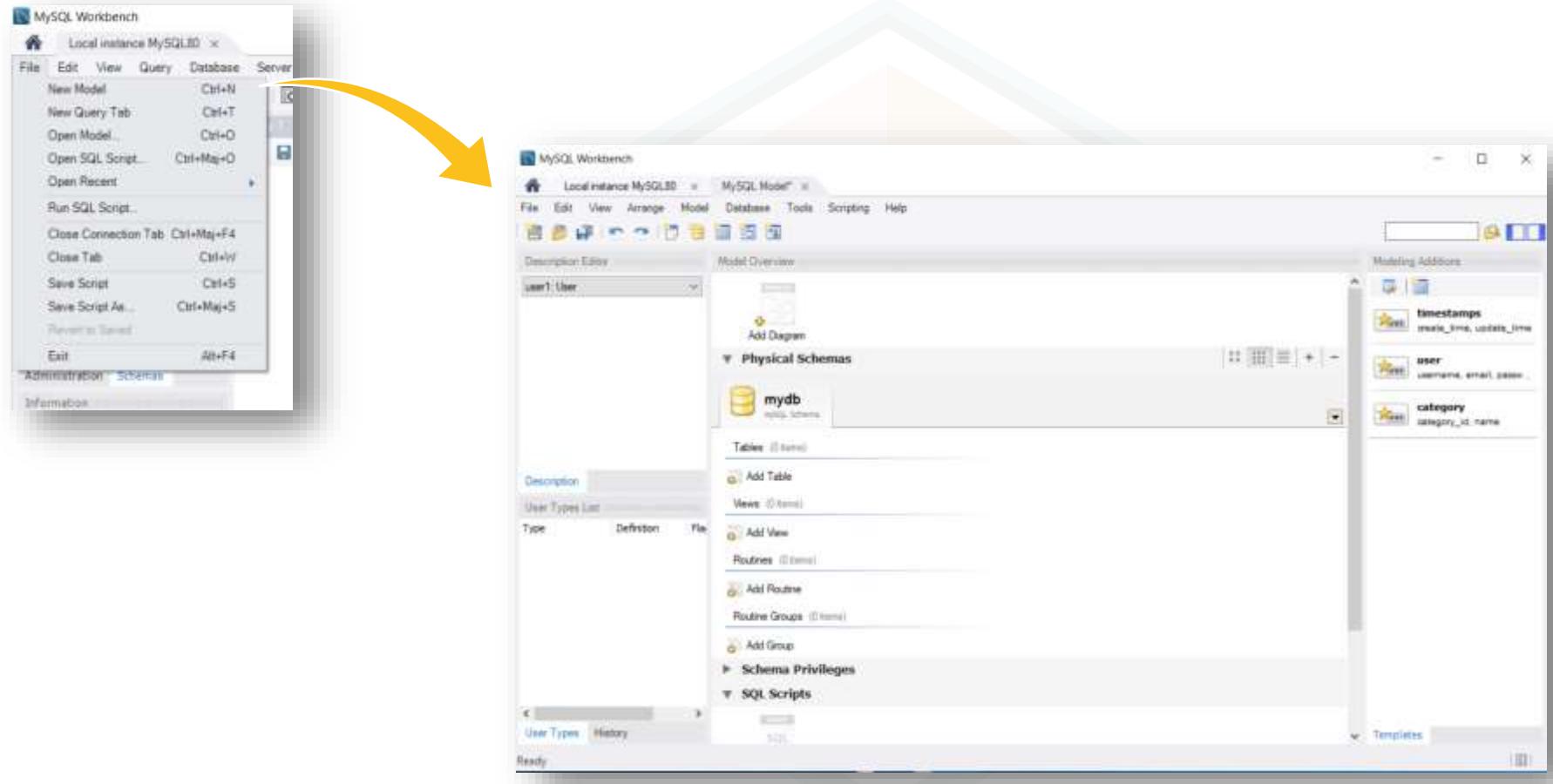
Création d'un utilisateur



Semifir

Création d'une base de données sur MySQL Workbench

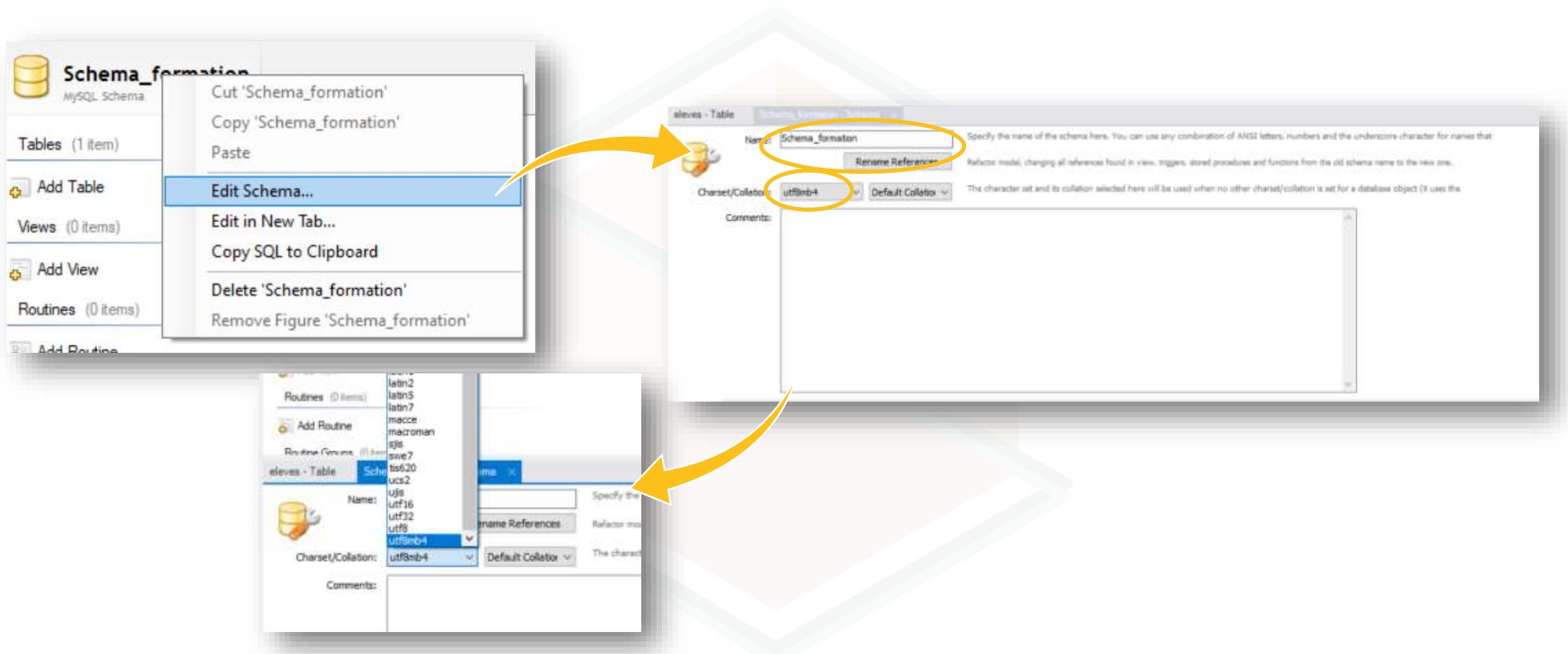
Création d'un modèle de base de données



Ouvrir un nouvel onglet à partir duquel nous allons pouvoir configurer notre base.

Création d'une base de données sur MySQL Workbench

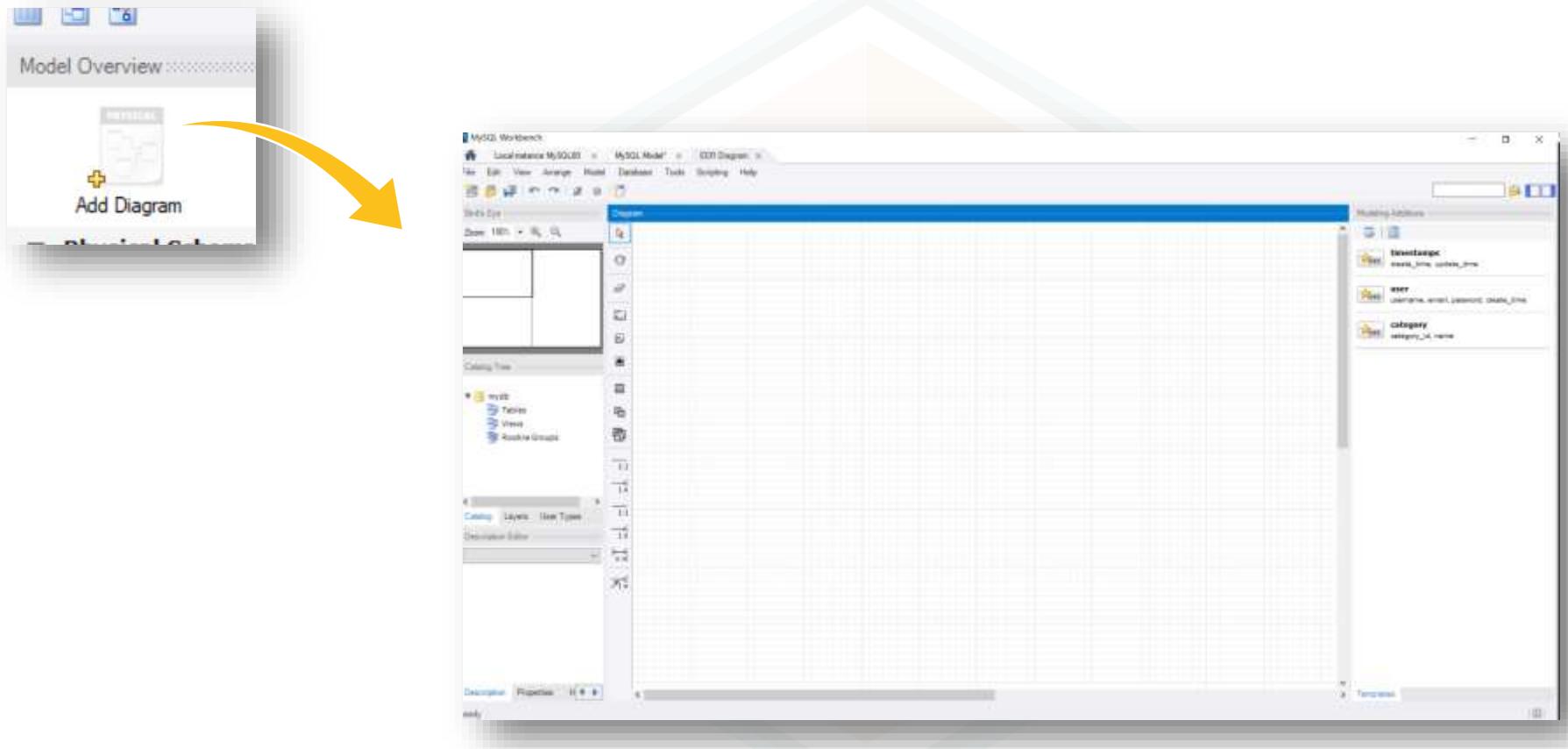
Création d'un modèle de base de données



'UTF8mb4' : correspond à la façon dont sont encodés les caractères (sur 8 bits, soit 2^8 caractères possibles, l'UTF8mb4 peut même stocker les emoji.

Création d'une base de données sur MySQL Workbench

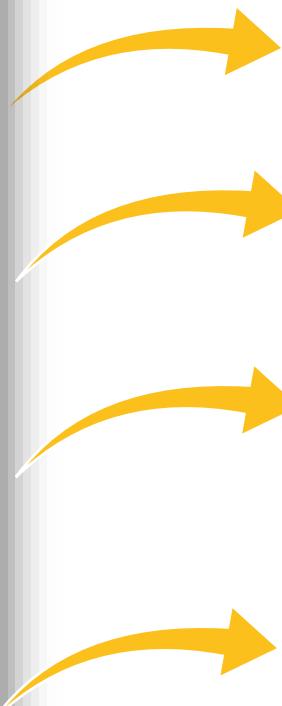
Vue diagramme :



Ouvrir un nouvel onglet à partir duquel nous allons pouvoir configurer notre base de manière graphique

Création d'une base de données sur MySQL Workbench

La barre d'outils :



Permet de sélectionner, déplacer ou supprimer un élément

***Permet de créer des éléments d'organisation purement visuels :
Layers, Notes et Images***

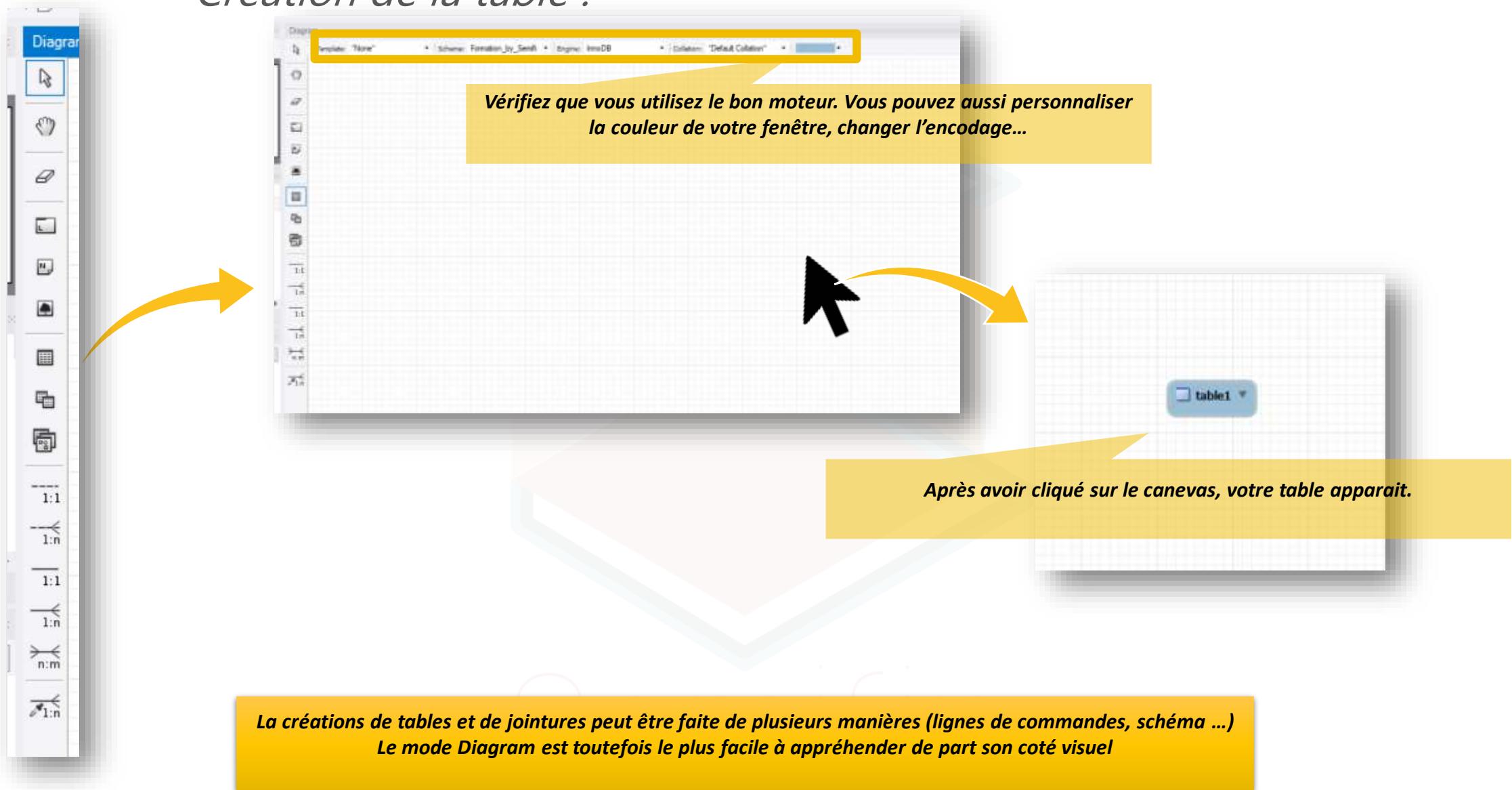
Permet de créer des Tables, Vues et Routines

Permet de créer des jointures entre les tables (1:1, 1:n, etc.)

*La créations de tables et de jointures peut être faite de plusieurs manières (lignes de commandes, schéma ...)
Le mode Diagram est toutefois le plus facile à appréhender de part son côté visuel*

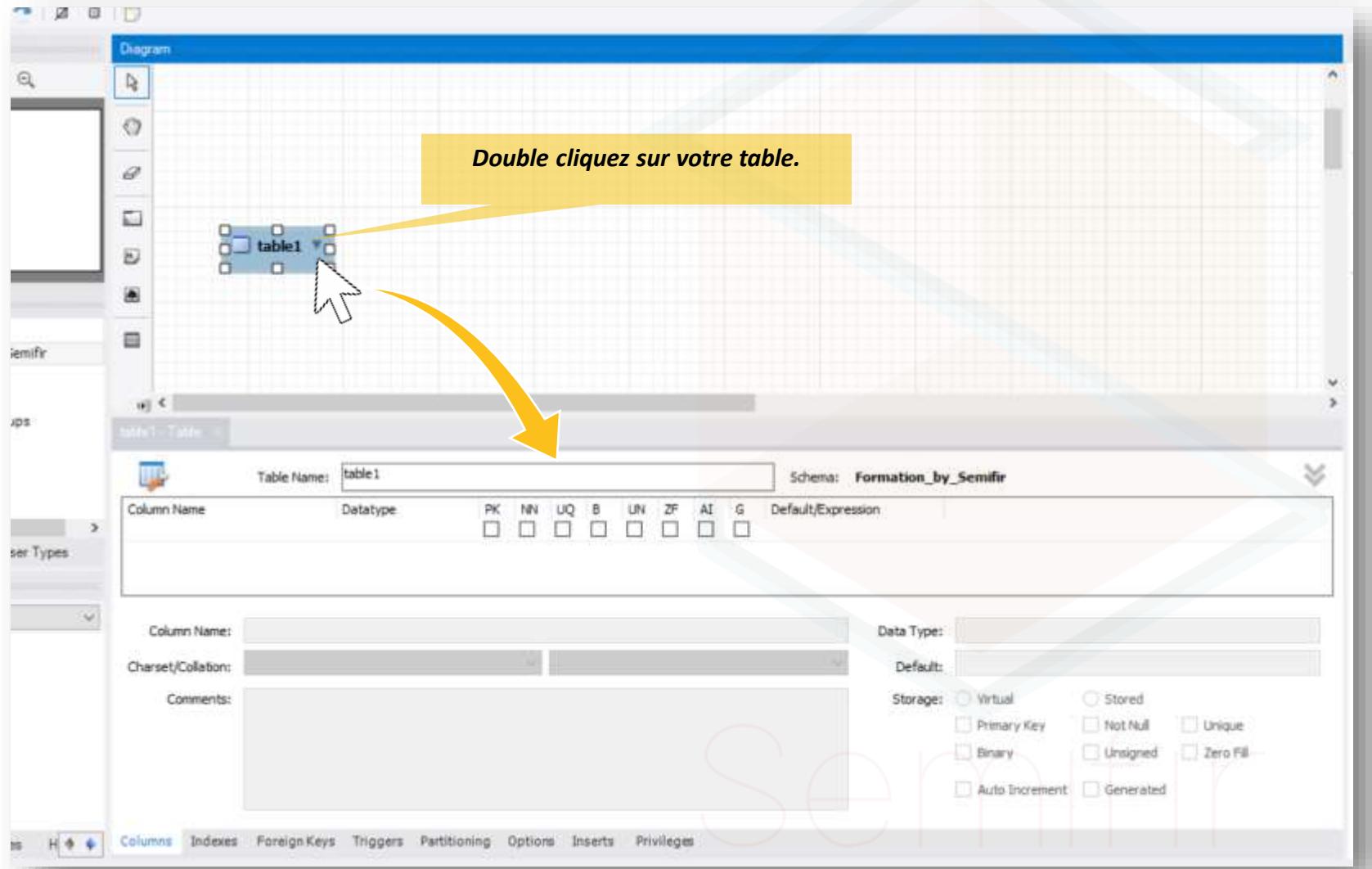
Création d'une base de données sur MySQL Workbench

Création de la table :



Création d'une base de données sur MySQL Workbench

Création de la table :



Création d'une base de données sur MySQL Workbench

Création d'un modèle de base de données

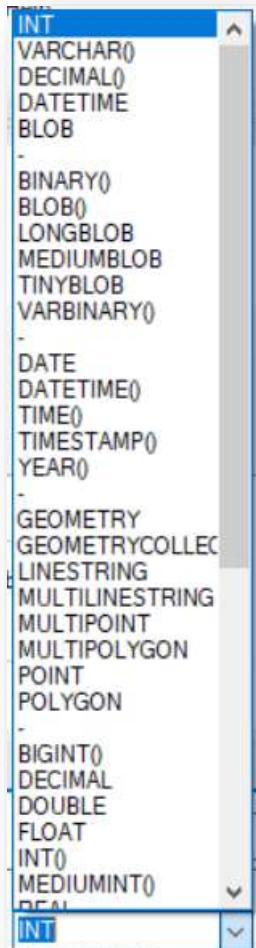
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
		<input type="checkbox"/>								

- **Column Name**
 - Contiendra le nom de notre colonne
- **Datatype**
 - Le type d'objet qui sera contenu dans la colonne.
 - Crée des restrictions quand au type et à la taille des objets que l'on pourra insérer.
- **Default/Expression**
 - Permet de saisir une valeur par défaut si la colonne n'est pas remplie lors de la création de la ligne
 - Permet aussi de saisir certains paramètres de colonnes, comme pour ENUM par exemple

- *Gardez en tête les conventions lors du nommage des colonnes !*
- *Pensez au contenu de vos colonnes pour définir le type d'objet utilisé !*

Création d'une base de données sur MySQL Workbench

Création d'un modèle de base de données



Concernant les DATATYPE

- Certains type (peu utilisés) n'ont pas été évoqués
 - Ils ne seront pas abordés dans ce cours car très spécifique
- Certains types suivis de () attendent une valeur
 - Ex : Varchar(45)
 - Dans le cas des STRING, il s'agit du nombre d'octets

Choisir le type d'objet

- Il est important de bien choisir le type d'objet d'une colonne
 - Une table mal conçue est un enfer à utiliser
 - Elle peut être modifiée plus tard certes, mais on peut facilement tout casser

- *Gardez en tête les conventions lors du nommage des colonnes !*
- *Pensez au contenu de vos colonnes pour définir le type d'objet utilisé !*

Création d'une base de données sur MySQL Workbench

Création d'un modèle de base de données

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
		<input type="checkbox"/>								

- UN : Unsigned
 - Indique un nombre qui sera toujours POSITIF
- ZF : Zero Fill
 - Remplit les caractères vides par des '0'. Permet d'avoir une utilisation mémoire fixe
- AI : Auto Increment
 - La seront automatiquement incrémentés lors de la création d'objet (nouvelle ligne)
- G : Generated
 - Permet de générer des valeurs via des fonctions et procédures

Création d'une base de données sur MySQL Workbench

Création d'un modèle de base de données

De quoi avons-nous besoin pour notre table ?

Demande : Créer une table pour gérer les notes des élèves

- Chaque élève doit avoir de renseigné son nom, prénom, date de naissance, adresse et sa classe
- Nous devons pouvoir créer un examen en précisant la matière, la date, l'intitulé de l'examen et un commentaire de l'instituteur

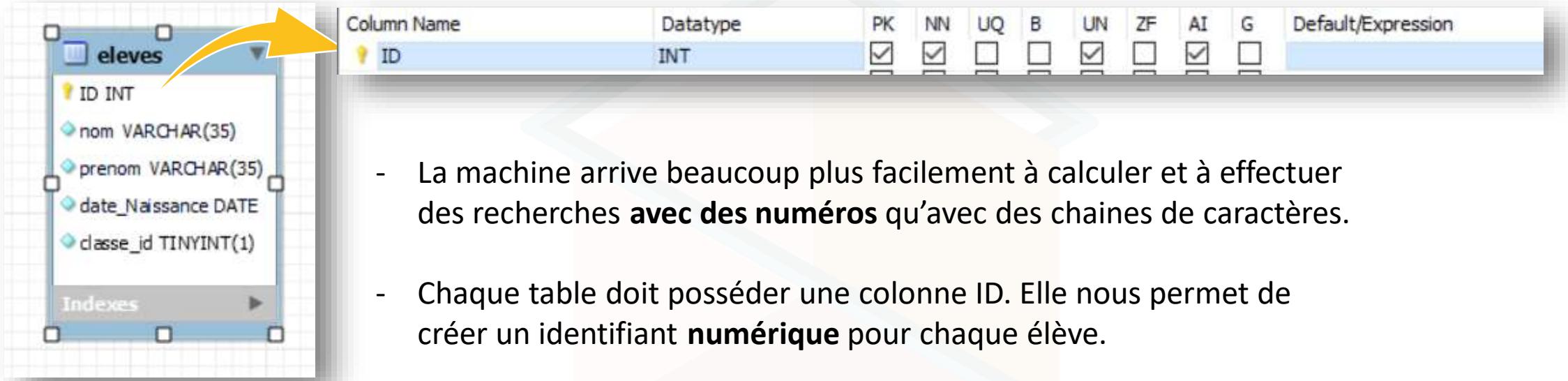
Commençons par la première table : La liste des élèves :



Let's beggin !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Colonne ID



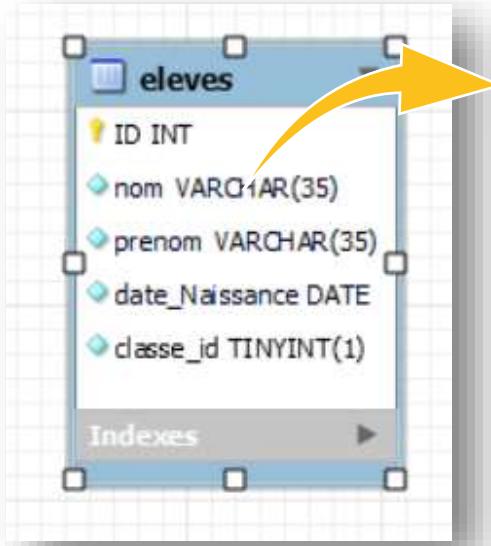
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
ID	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

- La machine arrive beaucoup plus facilement à calculer et à effectuer des recherches **avec des numéros** qu'avec des chaînes de caractères.
- Chaque table doit posséder une colonne ID. Elle nous permet de créer un identifiant **numérique** pour chaque élève.
- Les nombres en question seront **toujours positifs**.
- Chaque élève **DOIT** avoir un ID.
- Pour que les tables puissent communiquer efficacement entre elles, nous aurons besoin d'une **PRIMARY KEY** (1 par table max)

NB : ayez en tête les conventions !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Colonne nom



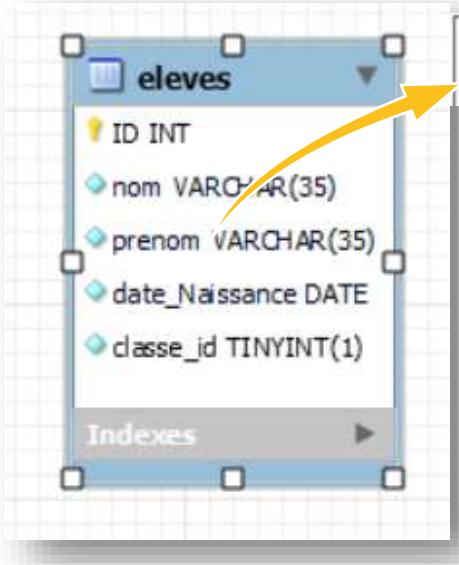
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
nom	VARCHAR(35)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						

- Nous aurons besoin d'une colonne 'nom'
- Chaque élève **DOIT** avoir un prénom.
- Un nom sera forcément une chaîne de caractères.
- Difficile d'estimer un nombre maximum, mais on peut supposer que personne n'a un prénom de plus de 35 caractères.

NB : ayez en tête les conventions !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Colonne prenom



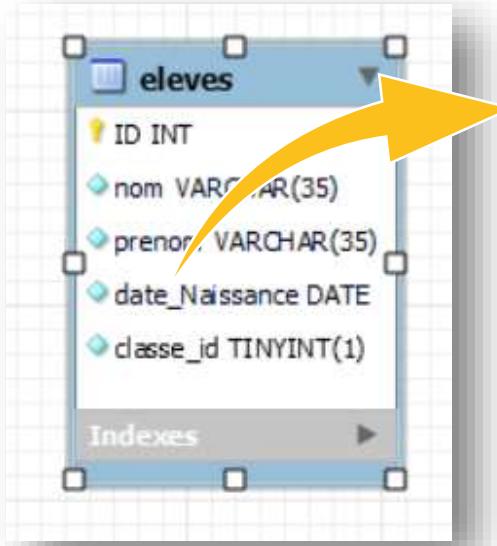
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
prenom	VARCHAR(35)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						

- Nous aurons besoin d'une colonne 'prenom'
- Chaque élève **DOIT** avoir un nom.
- Un nom sera forcément une chaîne de caractères.
- Difficile d'estimer un nombre maximum, mais on peut supposer que personne n'a un prénom de plus de 35 caractères.

NB : ayez en tête les conventions !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Colonne date_naissance

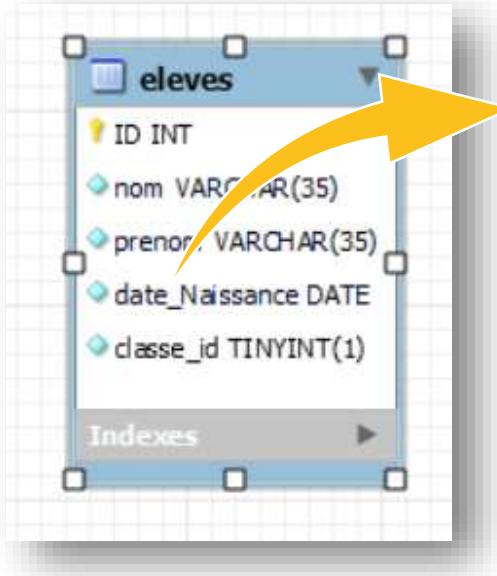


- Nous aurons besoin d'une colonne 'date_naissance'
- Chaque élève **DOIT** avoir une date de naissance.
- Une date de naissance ne peut être que de type **DATE**.

NB : ayez en tête les conventions !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Colonne classe_id



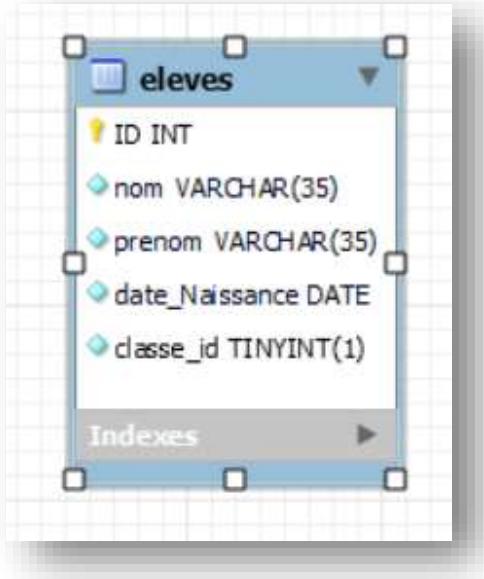
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
date_Naissance	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
classe_id	TINYINT(1)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

- Nous aurons besoin d'une colonne 'classe_id'
- L'idée étant d'utiliser des identifiants pour chacune de nos classes, en lieu et place du texte. L'ID nous permettra de croiser les tables facilement, d'éviter la duplication d'information, et d'accélérer les recherches.
- Un ID sera de type **INT**, avec **un seul chiffre** qui sera **toujours positif**

NB : ayez en tête les conventions !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Les contraintes



Les contraintes

- Toutes ces conditions ajoutées sur chaque colonnes sont appelées des contraintes.
- Elles permettent d'éviter qu'un utilisateur pas très malin ne rentre « topinambour » dans le champs « date_naissance »
- Elles garantissent l'uniformité des données
- Leur fonctionnement est similaire à un paramètre typé en POO

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Valider son diagramme et le transformer en BDD

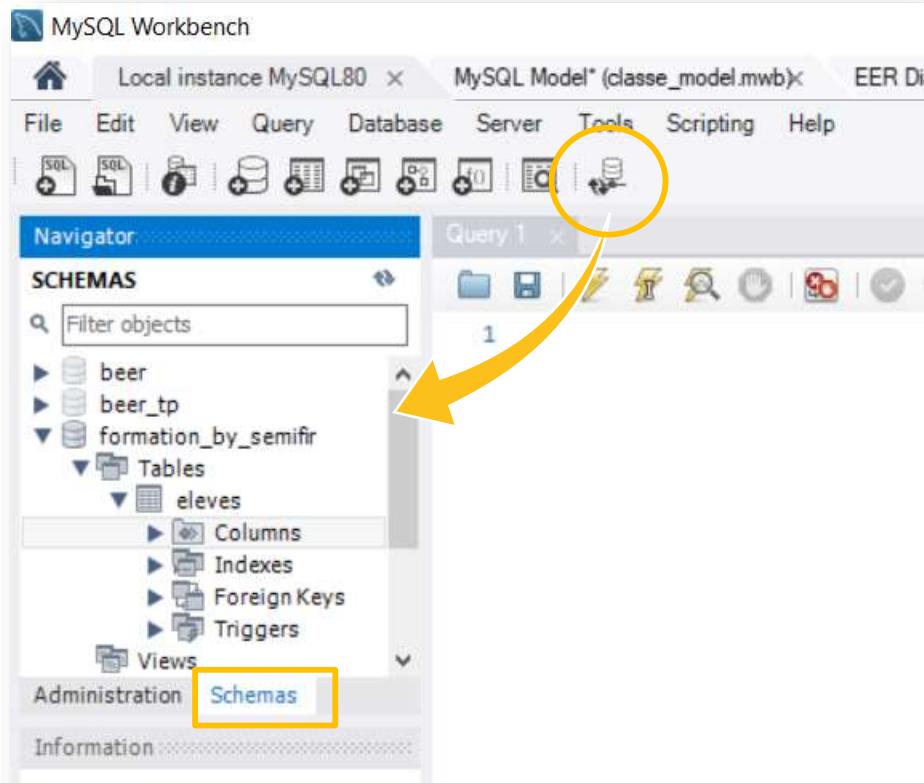


Cette fenêtre vous montre tout le code que vous n'avez pas eu à taper !
:D

Vous pouvez laisser par défaut et faire 'suivant'.

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Valider son diagramme et le transformer en BDD



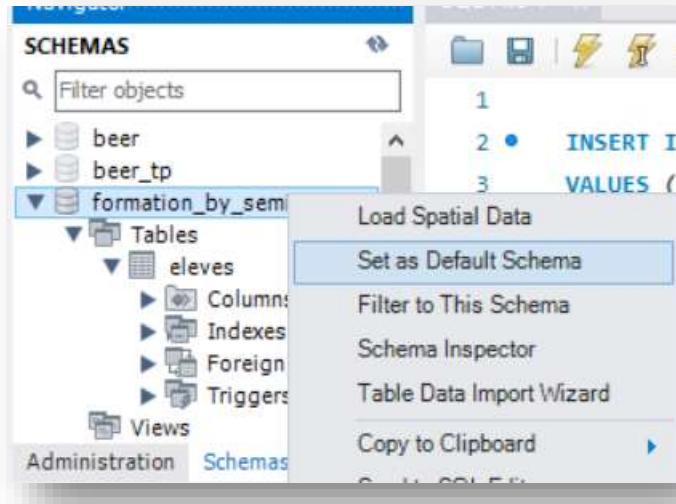
Notre base contient toutes les tables que nous avions rentré dans le diagramme.

Nous allons maintenant pouvoir peupler notre base de données avec nos premiers élèves !

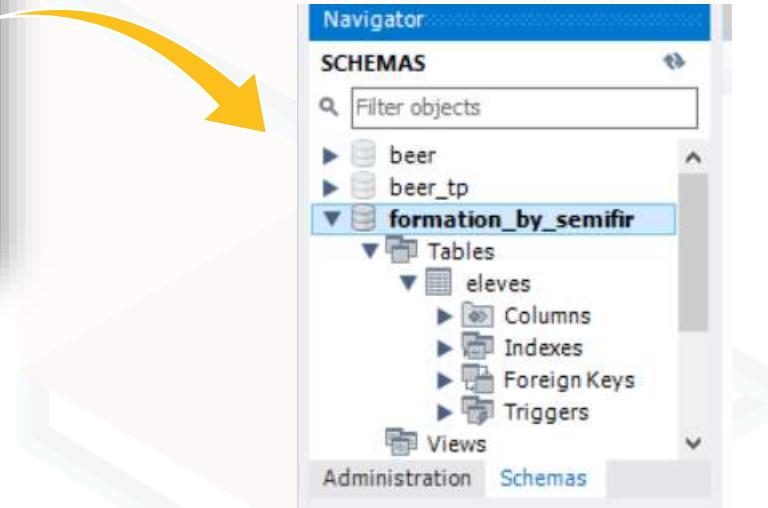
Après avoir actualisé la BDD, nous pouvons voir notre base 'Formation_by_semifir'

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Valider son diagramme et le transformer en BDD



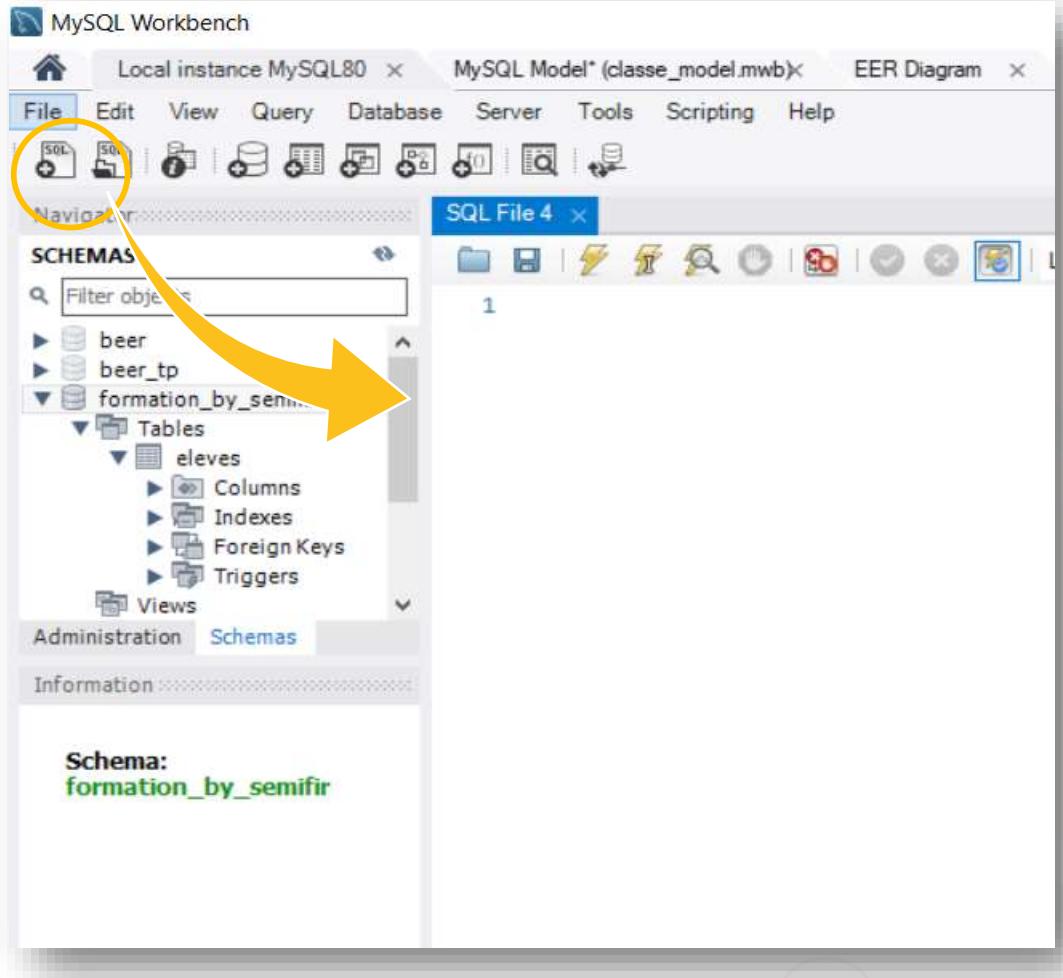
Une fois sélectionnée, la base apparaîtra en **gras**.



Il faut maintenant sélectionner notre base de données !

Création d'une base de données sur MySQL Workbench

Création de la table Elèves : Valider son diagramme et le transformer en BDD



Vous pouvez maintenant cliquer sur l'icone 'SQL+' pour ouvrir une nouvelle fenêtre de script. Nous allons pouvoir étudier les commandes de base !

Il est maintenant temps de peupler notre base <3

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Voici la syntaxe pour ajouter un élément :

```
INSERT INTO nom_table  
VALUES ('valeur_colonne_1'[, 'valeur_colonne2'[, 'valeur_colonne_3'[, etc.]])];
```

Donc dans notre cas :

```
INSERT INTO eleves  
VALUES (1, 'David', 'Bowie', '1947-01-08', 6);
```

Cette syntaxe implique que vous connaissiez l'ordre des colonnes, ainsi que l'état actuel de l'index !

N'oubliez pas de respecter les conventions !

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Voici la syntaxe pour lire les données d'une table :

```
SELECT
    colonne1[, -- Tapez "*" pour tout sélectionner
    colonne2[, etc.]
FROM
    nom_table;
```

*Par souci d'optimisation, il est fortement déconseillé d'utiliser « * » !
Préférez indiquer les colonnes dont vous avez besoin*

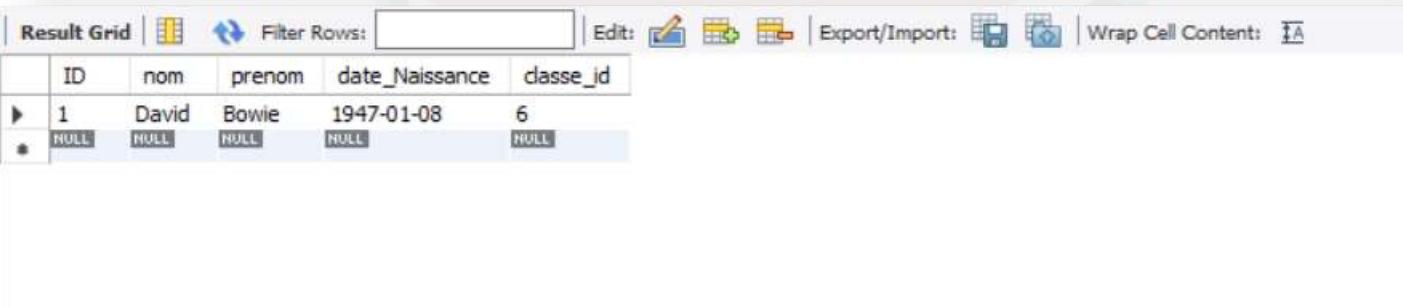
Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Donc dans notre cas :

```
SELECT *
FROM eleves;
```

Affichera un tableau contenant notre nouvel élève !



	ID	nom	prenom	date_Naissance	classe_id
▶	1	David	Bowie	1947-01-08	6
*	HULL	HULL	HULL	HULL	HULL

La dernière ligne indiquant 'NULL' sous entend que c'est la fin du tableau. Cette ligne n'existe 'pas vraiment' dans notre table.

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

```
SELECT  
    colonne1, colonne2, colonne3...  
FROM nom_table;
```

Exercice :

- Affichez uniquement les colonnes « nom » et « prénom » de nos élèves



	nom	prenom
▶	Bowie	David

*Par souci d'optimisation, il est fortement déconseillé d'utiliser « * » !
Préférez indiquer les colonnes dont vous avez besoin*

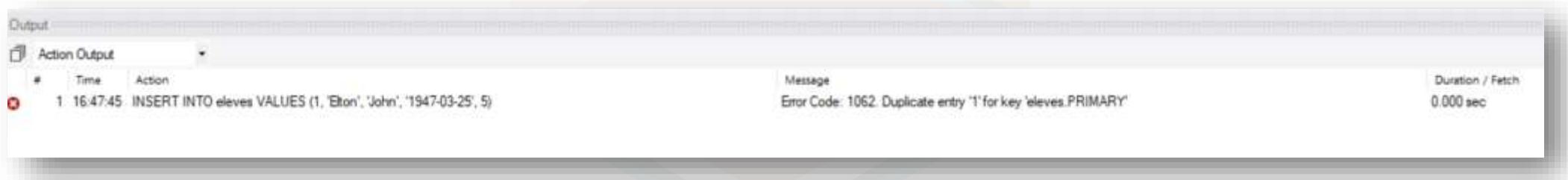
Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Ajoutons maintenant un second élève :

```
INSERT INTO eleves  
VALUES (1, 'Elton', 'John', '1947-03-25', 5);
```

Nous pouvons voir en bas que la machine nous a renvoyé une erreur !



L'erreur nous indique tout simplement que l'ID '1' est déjà utilisé !

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Plutôt que d'ajouter chaque élève 1 à 1, nous pouvons utiliser la syntaxe suivante :

```
INSERT INTO nom_table (colonne2, colonne5, colonne8)
-- On peut préciser les noms des colonnes que l'on souhaite remplir
-- Peu importe l'ordre tant que l'on remplit bien les valeurs
VALUES (data_c2, data_c5, data_c8);
-- Préciser les valeurs des colonnes
```

Cette syntaxe nous permet de sélectionner les colonnes à remplir. On peut donc choisir de ne pas préciser les colonnes auto incrémentées !

NB : n'oubliez pas d'entourer les valeurs de type string par des ''

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Il existe aussi une syntaxe alternative, propre à MySQL:

```
INSERT INTO nom_table  
SET colonne_1='data_c1', colonne_2='data_c2';
```

NB : cette syntaxe étant propre à MySQL, il est fortement déconseillé de l'utiliser !

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Ajoutons les élèves suivants, sans oublier notre petit Elton

```
INSERT INTO eleves (nom, prenom, date_Naissance, classe_id)
VALUES ('Elton', 'John', '1947-03-25', 6),
('Cartman', 'Éric', '1989-07-01', 7),
('Norton', 'Edward', '1959-08-18', 5),
('Nicks', 'Stevie', '1948-05-26', 4),
('Kilmister', 'Lemmy', '1945-12-24', 3);
```

*Ici, nous n'avons pas besoin de préciser l'ID : MySQL va l'auto
incrémenter pour nous <3*

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Vérifions maintenant que tout est OK :

```
SELECT  
*  
FROM  
eleves;
```

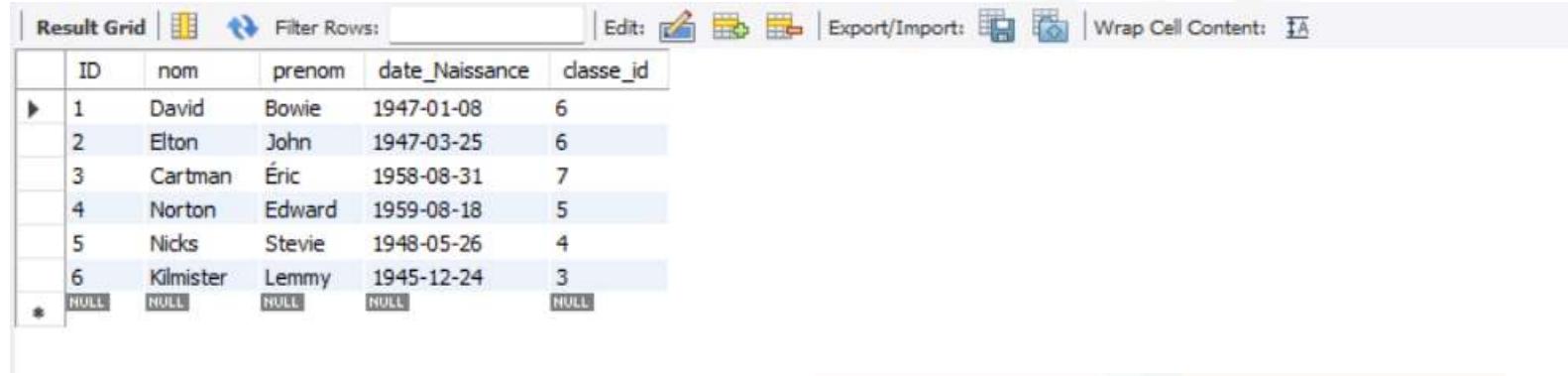


	ID	nom	prenom	date_Naissance	classe_id
▶	1	David	Bowie	1947-01-08	6
	2	Elton	John	1947-03-25	6
	3	Cartman	Éric	1958-08-31	7
	4	Norton	Edward	1959-08-18	5
	5	Nicks	Stevie	1948-05-26	4
	6	Kilmister	Lemmy	1945-12-24	3
*	HULL	NULL	NULL	NULL	NULL

La dernière ligne indiquant 'NULL' sous entend que c'est la fin du tableau. Cette ligne n'existe 'pas vraiment' dans notre table.

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données



ID	nom	prenom	date_Naissance	classe_id
1	David	Bowie	1947-01-08	6
2	Elton	John	1947-03-25	6
3	Cartman	Éric	1958-08-31	7
4	Norton	Edward	1959-08-18	5
5	Nicks	Stevie	1948-05-26	4
6	Kilmister	Lemmy	1945-12-24	3
*	HULL	HULL	HULL	HULL



Il semblerait qu'il y ait quelques petites erreurs :

- Le nom de notre BDD n'est pas cohérent,
- Le nom et prénom de David et Elton sont inversés,
- Une majuscule s'est glissée dans notre colonne 'date_Naissance'

La dernière ligne indiquant 'NULL' sous entend que c'est la fin du tableau. Cette ligne n'existe 'pas vraiment' dans notre table.

Création d'une base de données sur MySQL Workbench

Les commandes : Modifier des données

Voici la syntaxe pour modifier une colonne :

```
ALTER TABLE nom_table
    --Choisir la table à « altérer » (modifier)
    CHANGE ancien_nom nouveau_nom description_colonne;
    --Remplacer l'ancien nom par le nouveau EN REPRECISANT LA DESCRIPTION
```

Ici, « description » correspond aux contraintes.

Création d'une base de données sur MySQL Workbench

Les commandes : Modifier des données

Puisque nous devons saisir les contraintes (description), vérifions les options en question :

```
-- Syntaxe :  
DESCRIBE nom_table;  
  
-- Dans notre cas :  
DESCRIBE eleves;
```



Field	Type	Null	Key	Default	Extra
ID	int unsigned	NO	PRI	HULL	auto_increment
nom	varchar(35)	NO		HULL	
prenom	varchar(35)	NO		HULL	
date_Naissance	date	NO		HULL	
classe_id	tinyint unsigned	NO		HULL	

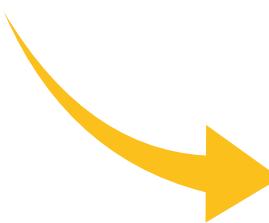
Avant de modifier quoi que ce soit, on VÉRIFIE CE QUE L'ON FAIT !

Création d'une base de données sur MySQL Workbench

Les commandes : Modifier des données

Donc, pour modifier la colonne « date_Naissance » :

```
ALTER TABLE eleves
--Choisir la table à altérer
CHANGE date_Naissance date_naissance DATE NOT NULL;
--Remplacer l'ancien nom par le nouveau EN REPRECISANT LA DESCRIPTION
```



	Field	Type	Null	Key	Default	Extra
▶	ID	int unsigned	NO	PRI	NULL	auto_increment
	nom	varchar(35)	NO		NULL	
	prenom	varchar(35)	NO		NULL	
	date_naissance	date	NO		NULL	
	classe_id	tinyint unsigned	NO		NULL	

Si vous souhaitez modifier uniquement les contraintes sans altérer le nom, il suffit alors de taper deux fois le même nom ;).

Création d'une base de données sur MySQL Workbench

Les commandes : Exporter des données

Au cas où les choses tourneraient mal, pensez à créer une sauvegarde de votre base !

La commande mysqldump ainsi que le workbench permettent de créer un fichier SQL contenant toute notre base.

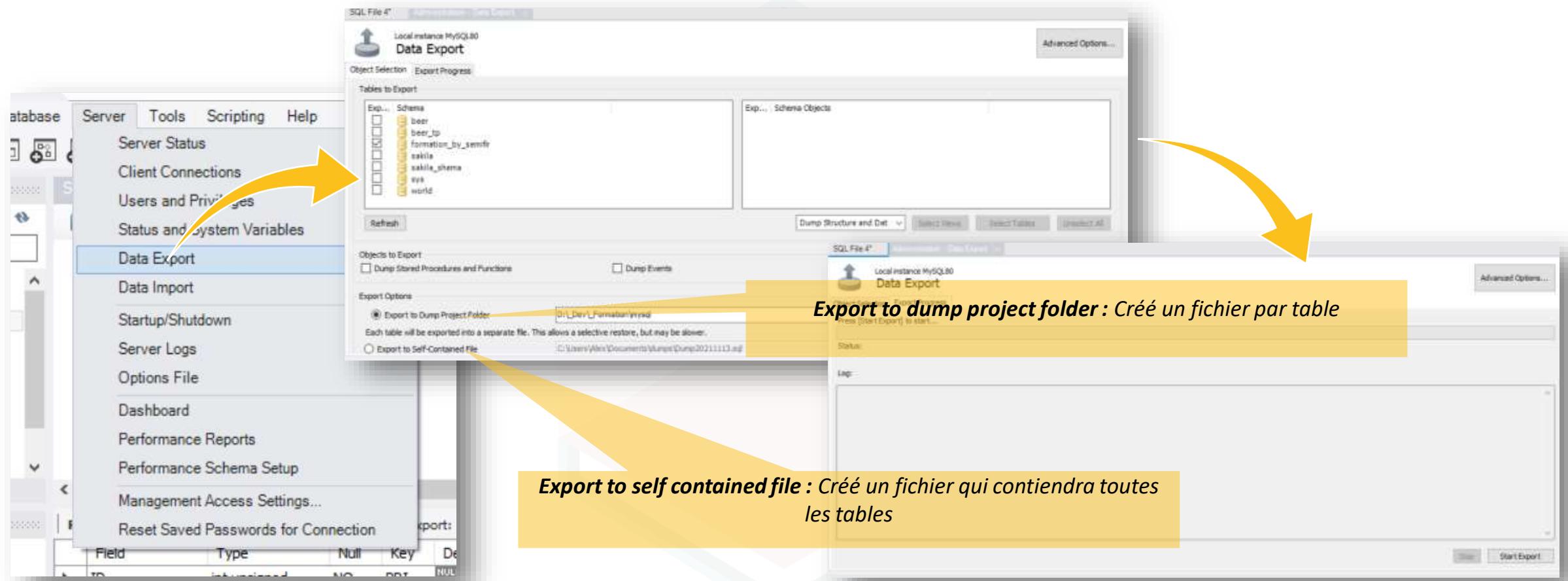
Le commande à taper :

```
mysqldump -u user -p --opt nom_base > nom_fichier.sql
```

Il est ceci dit bien plus simple dans notre cas d'utiliser l'option d'export du workbench

Création d'une base de données sur MySQL Workbench

Les commandes : Exporter des données



Avant toute modification de data, pensez à faire une sauvegarde !

Création d'une base de données sur MySQL Workbench

Les commandes : Exporter des données

Un petit coup d'œil à notre fichier :

```
-- MySQL dump 10.13 Distrib 8.0.27, for Win64 (x86_64)
-- Host: localhost Database: schema_formation
-- 
-- Server version 8.0.27

-- Table structure for table `eleves`
--

DROP TABLE IF EXISTS `eleves`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8mb4 */;
CREATE TABLE `eleves` (
  `ID` int unsigned NOT NULL AUTO_INCREMENT,
  `nom` varchar(35) COLLATE utf8_bin NOT NULL,
  `prenom` varchar(35) COLLATE utf8_bin NOT NULL,
  `date_naissance` date NOT NULL,
  `classe_id` tinyint unsigned NOT NULL,
  PRIMARY KEY ( `ID` )
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8mb3 COLLATE=utf8_bin;
/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table `eleves`
--

LOCK TABLES `eleves` WRITE;
/*!40000 ALTER TABLE `eleves` DISABLE KEYS */;
INSERT INTO `eleves` VALUES (1,'David','Bowie','1947-01-08',6),(2,'Elton','John','1947-03-25',6),(3,'Cartman','Eric','1989-07-01',7),(4,'Norton','Edward','1959-08-18',5),(5,'Nicks','Stevie','1948-05-26',4),(6,'Kilmister','Lemmy','1945-12-24',3);
/*!40000 ALTER TABLE `eleves` ENABLE KEYS */;
UNLOCK TABLES;
```

Le fichier ainsi créé contient toutes les informations pour recréer notre base et injecter les données via quelques commandes !

Création d'une base de données sur MySQL Workbench

Les commandes : Modifier des données

Syntaxe pour modifier un élément :

```
UPDATE nom_table  
SET colonne1= 'data1' [, colonne2 = 'data2', ...]  
-- implémente les données dans la/les colonnes ciblées...  
[WHERE condition]  
-- ... qui matchent la condition
```

Donc dans notre cas :

```
UPDATE eleves  
SET nom= 'Bowie' , prenom = 'David'  
WHERE nom LIKE 'David';
```

Mais d'où ça sort ces histoires de conditions ?

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Les conditions

Syntaxe pour filtrer un élément :

```
SELECT nom_colonne FROM nom_table  
WHERE [condition]  
ORDER BY [colonne ou options ASC DESC]
```

La commande **WHERE**, associée à un **SELECT** permet d'afficher uniquement les objets qui matchent notre condition.

ORDER BY permet de choisir comment les données seront triées (**ASC** pour croissant, **DESC** pour décroissant)

Pour expliciter les conditions, nous utilisons des opérateurs de comparaison, des opérateurs logiques, ainsi que d'autres commandes comme **LIKE**.

WHERE est donc une condition (clause) de SELECT

Création d'une base de données sur MySQL Workbench

Les commandes : Afficher des données

Exercice

```
SELECT colonnes  
FROM table  
ORDER BY
```

Triez les élèves en fonction de leur date de naissance.

N'affichez que le nom, prénom et date de naissance



	nom	prenom	date_naissance
▶	Kilmister	Lemmy	1945-12-24
	Bowie	David	1947-01-08
	Elton	John	1947-03-25
	Nicks	Stevie	1948-05-26
	Norton	Edward	1959-08-18
	Cartman	Eric	1989-07-01

Création d'une base de données sur MySQL Workbench

Les commandes : Modifier des données

```
UPDATE nom_table  
SET colonne1= 'data1'  
[WHERE condition]
```

Exercice :

Modifiez l'élève Elton John : remettez son nom et son prénom à leur place .

Semifir

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Les opérateurs

Opérateurs de comparaison :

Opérateur	Signification
=	Égal
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
<> ou !=	Différent
<=>	Égal (pour NULL)

Opérateurs logiques :

Opérateur	Symbole	Signification
AND	&&	Et
OR		Ou
XOR		Ou exclusif
NOT	!	Non/pas

On peut utiliser soit l'opérateur logique, soit son symbole

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Syntaxe pour filtrer un élément sur plusieurs critères :

```
SELECT * FROM nom_table  
WHERE colonne1='data1'  
      OR colonne1 = 'data2';
```

On a donc deux conditions : on recherche l'un OU l'autre

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Syntaxe pour filtrer un élément sur plusieurs critères, un peu plus complexe :

```
SELECT *
FROM nom_table
WHERE colonne1 > 'data1'
    OR
    ( colonne2='data2'
        AND
        ( colonne3='data3'
            OR
            ( colonne3='data4' AND colonne1 < 'data5')
        )
    );

```

Nous pouvons donc chainer les critères grâce aux parenthèses

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

LIKE et les recherches approximatives :

```
SELECT * FROM nom_table  
WHERE nom_colonne LIKE '%exemple%';
```

Les jokers de **LIKE** :

- ‘_’ : Représente UN SEUL caractère
- ‘%’ : Représente un ou plusieurs caractères

Les jokers peuvent être associés et mélangés

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Exemples :

- 'a%': recherchera une chaîne qui commence par 'a' (allons, aller, amis ...)
- 'd_': recherchera une chaîne qui commence par 'd' suivi d'une lettre (de, da, du ...)
- 'b__e' : recherchera une chaîne qui commence par 'b', suivi de 3 lettres et qui se termine par 'e' (bowie, balle)
- %es% : recherchera un string qui contient 'es'

Si les caractères '%' ou '_' font partie de votre recherche, pensez au caractère d'échappement '\'

Création d'une base de données sur MySQL Workbench

Les commandes : Filtrer des données

Précisions :

- **LIKE** peut être utilisé avec les objets de type **INT**
- **LIKE** peut être utilisé en combinaison avec **NOT**
- On peut utiliser l'option **BINARY** pour rendre **LIKE** case sensitif
- Si les caractères '%' ou '_' font partie de votre recherche, pensez à utiliser le caractère d'échappement '\'

Création d'une base de données sur MySQL Workbench

Les commandes : Filtrer des données

```
SELECT * FROM nom_table  
WHERE colonne1='data1'  
      OR (colonne1 = 'data2' AND colonne1 = 'data3');
```

Exercice :

Filtrer les élèves selon les critères suivants :

- Nés après 1945,
- Dont le nom ne contient pas de 'a' ET pas de 'k'
- Dont le prénom contient un 'n'
- Affichez le nom, prénom et date de naissance.

Création d'une base de données sur MySQL Workbench

Les commandes : Filtrer des données

Exemple:

```
SELECT nom, prenom, date_naissance  
FROM eleves  
WHERE date_naissance > '1945-12-31'  
      AND (nom NOT LIKE '%a%' AND nom NOT LIKE '%k%')  
      AND prenom LIKE '%n%'
```

On affiche le nom et prénom des élèves...

... qui sont nés après 1945

... ET dont le nom ne contient ni 'a' ni 'k'

... ET dont le prénom contient 'n'

Ca a l'air compliqué, mais non !

	nom	prenom	date_naissance
▶	Elton	John	1947-03-25

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Les recherches d'intervalles :

```
SELECT * FROM nom_table  
WHERE colonne BETWEEN 'data1' AND 'data2';
```

Exercice :

Affichez les élèves nés entre le 01/01/1945 et le 31/12/1947

N'affichez que le nom, prénom et date de naissance

- Fonctionne avec des INT,
- Possible d'ajouter NOT pour exclure l'intervalle
- Possible d'y ajouter BINARY (case sensitif)

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Exemple:

```
SELECT
    nom, prenom, date_naissance
FROM
    eleves
WHERE
    date_naissance BETWEEN '1945-01-01' AND '1947-12-31'
```



	nom	prenom	date_naissance
▶	Bowie	David	1947-01-08
	Elton	John	1947-03-25
	Kilmister	Lemmy	1945-12-24

Ca a l'air compliqué, mais non !

Création d'une base de données sur MySQL Workbench

Les commandes : *Filtrer des données*

Les critères multiples :

```
SELECT * FROM nom_table  
WHERE colonne IN ('critère1', 'critère2', 'etc.');
```

Exercice :

Utilisez cette commande pour n'afficher QUE les élèves de
6^{ème} et 5^{ème}

N'affichez que le nom, le prénom et la classe.

- Pratique pour éviter de chainer les 'AND'
- Possible d'ajouter NOT

Création d'une base de données sur MySQL Workbench

Les commandes : Filtrer des données

Exemple:

```
SELECT
    nom, prenom, classe_id
FROM
    eleves
WHERE classe_id IN ('6', '5')
```

Revient à écrire :

```
SELECT
    nom, prenom, classe_id
FROM
    eleves
WHERE classe_id LIKE '6' OR classe_id LIKE '5'
```



	nom	prenom	classe_id
▶	Bowie	David	6
	Elton	John	6
	Norton	Edward	5

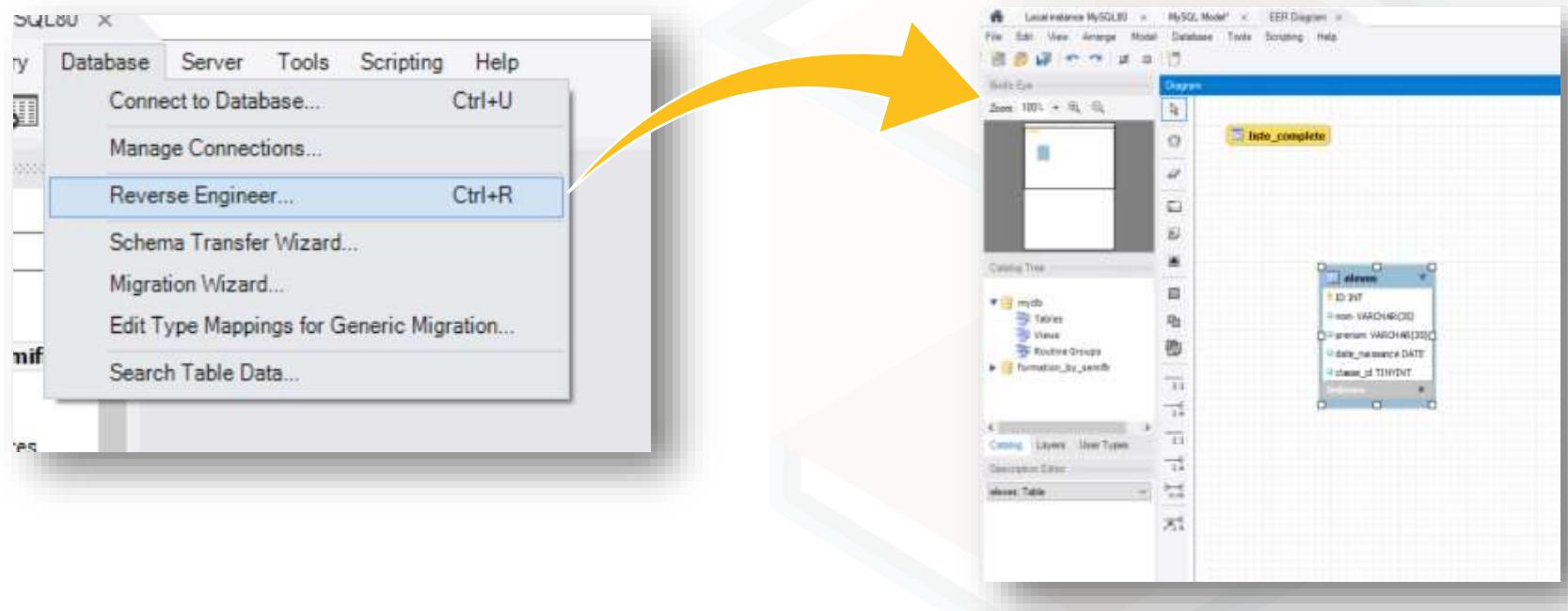


C'est quand même plus lisible ☺

Création d'une base de données sur MySQL Workbench

MySQL Workbench : Création d'une seconde table

Retournons maintenant sur notre diagramme afin de créer une seconde table :



Pensez à bien sélectionnez la bonne base de données avant de poursuivre.

Création d'une base de données sur MySQL Workbench

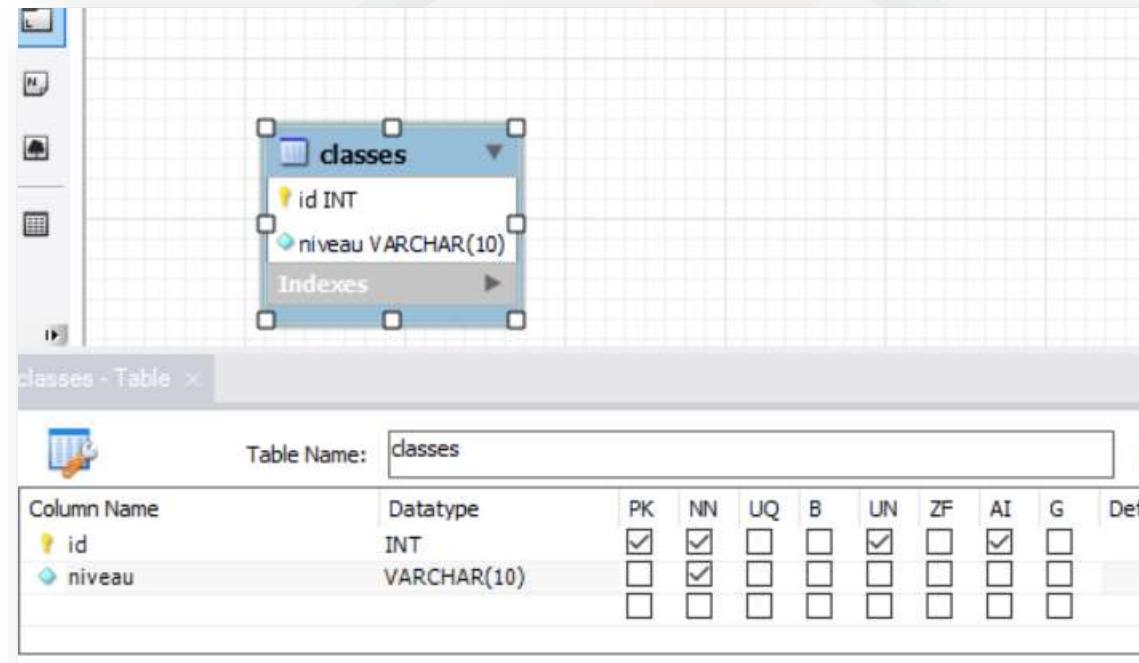
MySQL Workbench : Création d'une seconde table

Créons la table « classes » avec les paramètres suivants :

- Il y a 5 classes, de la classe de sixième à la troisième ainsi que la classe segpa.
 - Quel type utiliser ?
 - Comment appeler la colonne ?
- Chaque classe doit posséder un ID
 - Quel type et paramètres utiliser ?

Création d'une base de données sur MySQL Workbench

MySQL Workbench : Création d'une seconde table



Voici à quoi ressemble notre seconde table

4. Les jointures



Les jointures

MySQL Workbench : Relier les bases entre elles

Les clefs primaires :

- Chaque table peut disposer d'une seule clef primaire,
- Son utilisation n'est pas obligatoire mais c'est vivement recommandé,
- Elle permet d'identifier chaque ligne de la table de manière unique
- S'agissant d'un index, elle permet d'accélérer la rapidité des recherches

Elle doit :

- Être neutre vis-à-vis des données
 - Elle ne peut être éditée une fois saisie,
 - Pour la modifier, il faudrait supprimer l'élément et le recréer
- Être unique
- Être toujours positive

BEST PRACTICE

Les jointures

MySQL Workbench : Relier les bases entre elles

Ici, l'ID est notre clef primaire

ID	Nom	Prénom	Date de naissance	Classe
1	S Bowie	David	08/01/1947	6
2	Elton	John	25/03/1947	6
3	Cartman	Éric	01/07/1989	7
4	Jorton	Edward	18/08/1969	5
5	Jicks	Stevie	26/05/1948	4
6	Gilmister	Lemmy	24/12/1945	3

Nous avons déjà créé la clef primaire lors de la création de notre table

Les jointures

MySQL Workbench : Relier les bases entre elles

Les clefs étrangères :

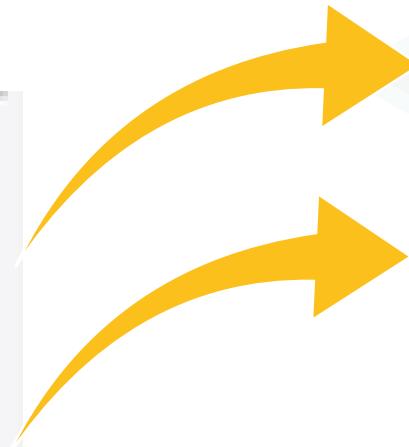
- Permettent de garantir l'uniformité des données inter-tables,
- Correspondent à la clef primaire d'une autre table
 - Permet donc de les mettre en relation

S'agissant de la clef primaire d'une autre table, elle est soumise aux même règles.

Faute de clef étrangère, nous ne pourrons pas faire communiquer nos tables

Les jointures

MySQL Workbench : Création d'une seconde table



Jointure 'One to One' :

- L'élément de la table 'A' ne peut correspondre qu'à un seul élément de la table 'B'

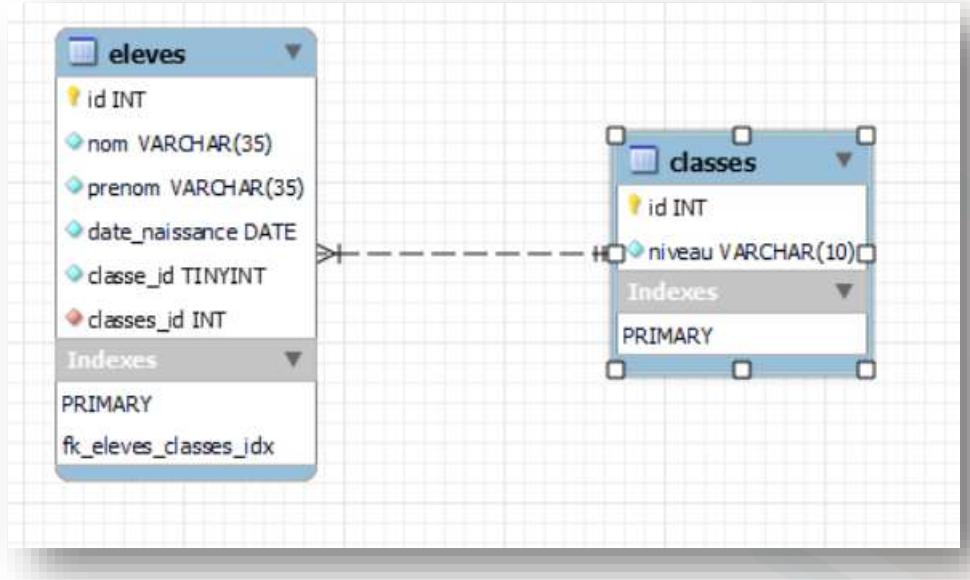
Jointure 'One to Many' :

- L'élément de la table 'A' peut correspondre à plusieurs éléments de la table 'B'

1:1 or 1:n ?

Les jointures

MySQL Workbench : Création d'une seconde table



Dans notre cas :

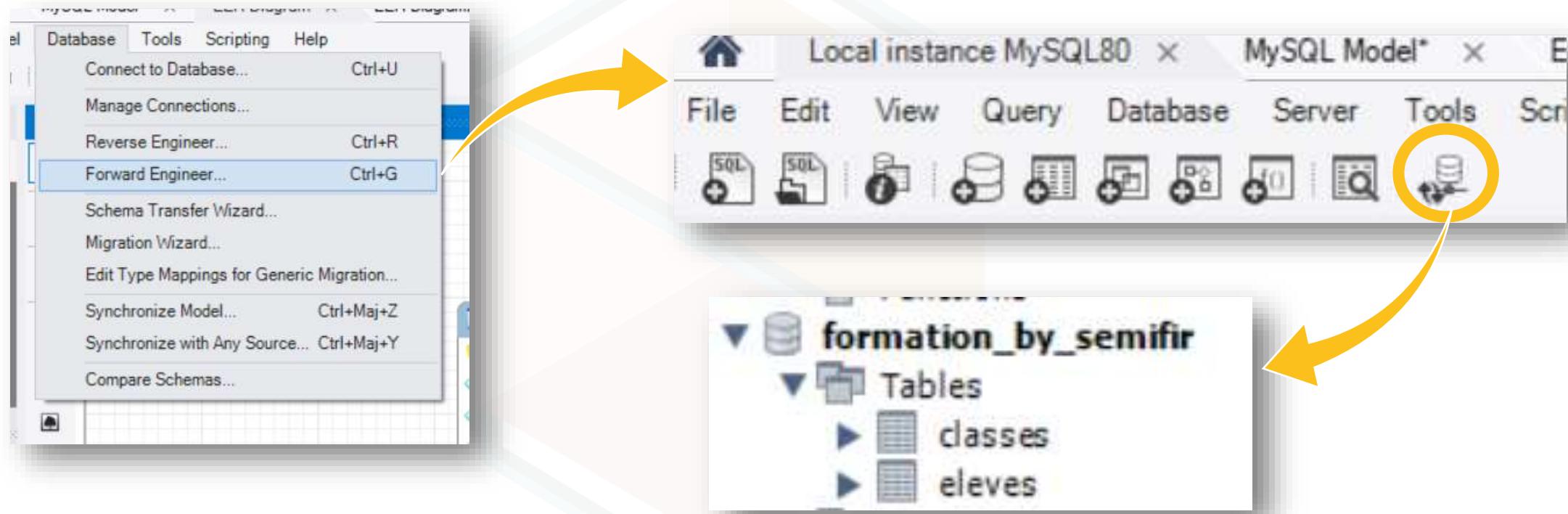
- Une classe peut accueillir plusieurs élèves,
- Un élève ne peut être que dans une seule classe

En créant notre clef étrangère, nous avons une nouvelle ‘colonne’ qui est apparue dans la liste : c'est notre clef étrangère,
Il est maintenant temps de la remplir

Nous avons maintenant créé notre jointure !

Les jointures

MySQL Workbench : Création d'une seconde table



N'oubliez pas d'actualiser votre base !

Les jointures

Les clefs : Création d'une seconde table

Il est temps de remplir notre table « classes » :

```
INSERT INTO classes (id, niveau)
VALUES (7, 'segpa'),
(6, 'Sixième'),
(5, 'cinquième'),
(4, 'Quatrième'),
(3, 'Troisième');
```

```
SELECT
*
FROM
classes
```

Result Grid	
id	niveau
3	Troisième
4	Quatrième
5	cinquième
6	Sixième
7	SEGPA
NULL	NULL

*Par souci de simplicité, nous faisons correspondre les ID et le niveau.
Également, nos forçons l'ID*

Introduction aux bases de données

Table classe

ID	Niveau
6	Sixième
5	Cinquième
4	Quatrième
3	Troisième
7	Segpa

Table élèves :

ID	Nom	Prénom	Date de naissance	Classe_ID
1	Bowie	David	08/01/1947	6 (Sixième)
2	John	Elton	25/03/1947	6 (Sixième)
3	Cartman	Éric	01/07/1989	7 (Segpa)
4	Norton	Edward	18/08/1969	5 (Cinquième)
5	Nicks	Stevie	26/05/1948	4 (Quatrième)
6	Kilmister	Lemmy	24/12/1945	3 (Troisième)

Notre objectif est de faire correspondre nos tables

Les jointures

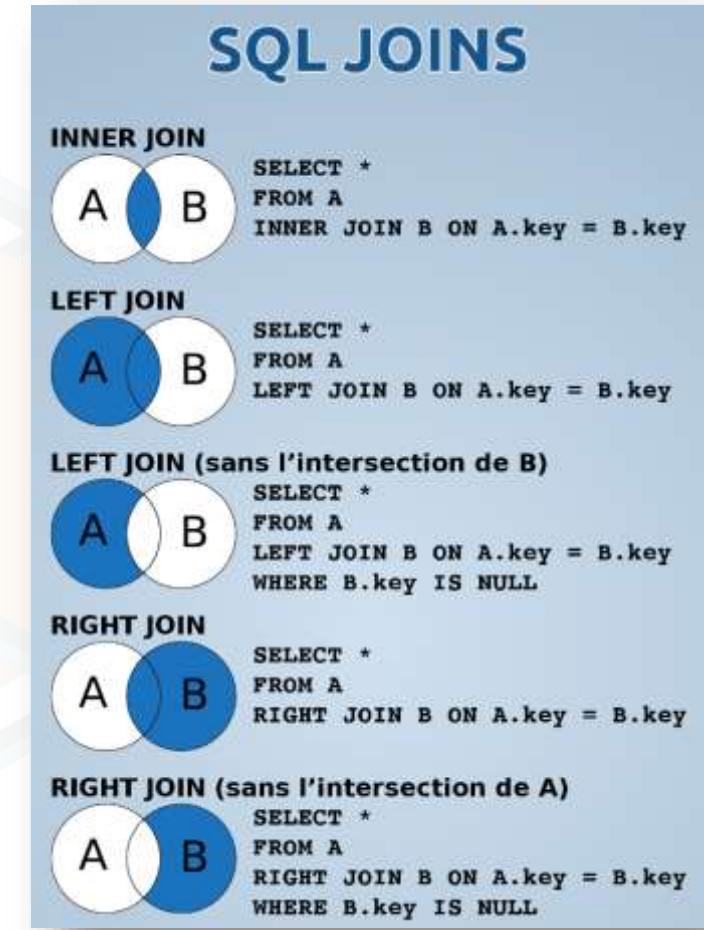
Définition :

La jointure nous permet :

- De lire les informations inter-tables
- D'afficher les informations
 - Ex : Afficher le niveau de la classe au lieu de son ID

Types de jointures :

- Les jointures naturelles,
- Les jointures par la gauche,
- Les jointures par la droite,
- Les jointures internes.



On vérifie rapidement que tout est OK

Les jointures

Création d'une seconde table

Modifions notre base pour illustrer les jointures :

Ajout d'un nouveau niveau dans la table « classe », dans lequel il n'y a aucun élève

```
INSERT INTO classes  
(id, niveau)  
VALUE (2, 'Seconde')
```

Ajout d'un élève avec une classe_id qui n'existe pas dans 'classes' (8)

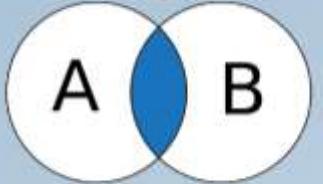
```
INSERT INTO eleves (nom, prenom, date_naissance, classe_id)  
VALUES ('Hudson', 'Saul', '1965-07-23', 8)
```

Saul Hudson : 'Slash' (Guns N' roses)

Les jointures

INNER JOIN

INNER JOIN



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

INNER JOIN :

Permet de renvoyer l'information quand cette dernière est **vraie dans les deux tables**.

C'est la jointure la plus commune

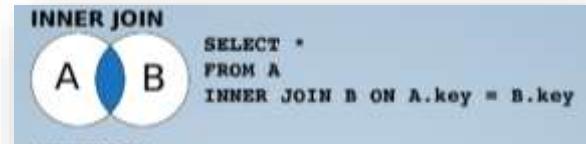
```
SELECT * FROM nom_table_1  
[INNER] JOIN nom_table_2  
    ON colonne_table1 = colonne_table2  
[WHERE ...]  
[ORDER BY ...]  
[LIMIT ...]
```

- On peut préciser les colonnes que l'on souhaite afficher. Elles peuvent être présentes sur une table ou sur l'autre mais il faut préciser son origine
- Si non précisé, INNER est implicite et par défaut
- 'ON' sert à indiquer quelles colonnes doivent correspondre.

INNER JOIN est le plus souvent utilisé

Les jointures

INNER JOIN



```
SELECT *
FROM eleves
INNER JOIN
    classes ON classe_id = classes.id
ORDER BY classe_id
```

Le nom de la seconde table

*Le nom de la colonne de
« eleves »*

Le nom de notre « table2.la_colonne »

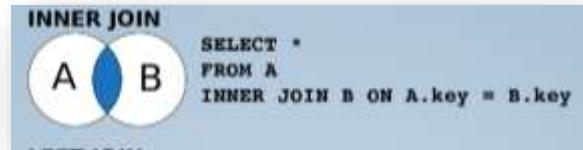
Exemple pas très BEST PRACTICE

A screenshot of a database result grid titled "Result Grid". The grid has columns: ID, nom, prenom, date_naissance, classe_id, id, and niveau. The data is as follows:

ID	nom	prenom	date_naissance	classe_id	id	niveau
6	Kilmister	Lemmy	1945-12-24	3	3	Troisième
5	Nicks	Stevie	1948-05-26	4	4	Quatrième
4	Norton	Edward	1959-08-18	5	5	cinquième
1	Bowie	David	1947-01-08	6	6	Sixième
2	John	Elton	1947-03-25	6	6	Sixième
3	Cartman	Éric	1989-07-01	7	7	secpa

Les jointures

INNER JOIN



✓ Vérifie si chaque ligne de la table 1 a une correspondance sur la table 2

ID	nom	prenom	date_naissance	classe_id
1	Bowie	David	1947-01-08	6
2	John	Elton	1947-03-25	6
3	Cartman	Éric	1989-07-01	7
4	Norton	Edward	1959-08-18	5
5	Nicks	Stevie	1948-05-26	4
6	Kilmister	Lemmy	1945-12-24	3
*	NULL	NULL	NULL	NULL

Avec cette méthode, seront affichées les correspondances entre chacune de nos tables, et ce pour chaque ligne !

	id	niveau
▶	2	Seconde
	3	Troisième
	4	Quatrième
	5	cinquième
	6	Sixième
	7	segpa
*	NULL	NULL

ID	nom	prenom	date_naissance	classe_id	id	niveau
6	Kilmister	Lemmy	1945-12-24	3	3	Troisième
5	Nicks	Stevie	1948-05-26	4	4	Quatrième
4	Norton	Edward	1959-08-18	5	5	cinquième
1	Bowie	David	1947-01-08	6	6	Sixième
2	John	Elton	1947-03-25	6	6	Sixième
3	Cartman	Éric	1989-07-01	7	7	segpa

La commande a concaténé nos deux tableaux, et créé une « table virtuelle »

Les jointures

INNER JOIN

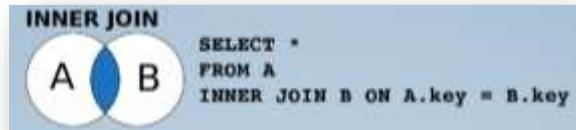
'AS' permet de définir un alias. Dans notre cas, il bonifie l'affichage

```
SELECT
    nom AS Nom,
    prenom AS Prénom,
    date_naissance AS 'Date de naissance',
    classes.niveau AS Classe
FROM
    eleves
        INNER JOIN
    classes ON classe_id = classes.id
ORDER BY classe_id
```



Nom	Prénom	Date de naissance	Classe
Kilmister	Lemmy	1945-12-24	Troisième
Nicks	Stevie	1948-05-26	Quatrième
Norton	Edward	1959-08-18	cinquième
Bowie	David	1947-01-08	Sixième
John	Elton	1947-03-25	Sixième
Cartman	Éric	1989-07-01	segpa

Affichons quelque chose de plus propre



Les jointures

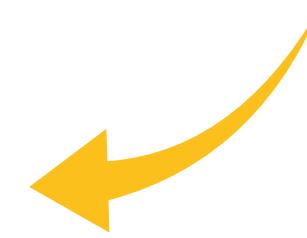
INNER JOIN

ID	nom	prenom	date_naissance	classe_id
1	Bowie	David	1947-01-08	6
2	John	Elton	1947-03-25	6
3	Cartman	Éric	1989-07-01	7
4	Norton	Edward	1959-08-18	5
5	Nicks	Stevie	1948-05-26	4
6	Kilmister	Lemmy	1945-12-24	3
7	Hudson	Saul	1965-07-23	8



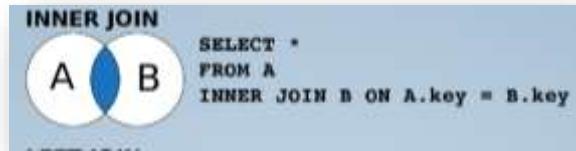
	Nom	Prénom	Date de naissance	Classe
▶	Kilmister	Lemmy	1945-12-24	Troisième
	Nicks	Stevie	1948-05-26	Quatrième
	Norton	Edward	1959-08-18	cinquième
	Bowie	David	1947-01-08	Sixième
	John	Elton	1947-03-25	Sixième
	Cartman	Éric	1989-07-01	segpa

	id	niveau
▶	2	Seconde
	3	Troisième
	4	Quatrième
	5	cinquième
	6	Sixième
	7	segpa
*	NULL	NULL



Mais il est où Slash ? oO

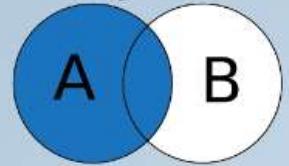
La classe ID « 8 » n'étant pas présente dans les deux tables, Slash n'a pas été repris : la condition n'étant pas vraie des deux côtés !



Les jointures

LEFT JOIN

LEFT JOIN



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

```
SELECT * FROM nom_table  
LEFT [OUTER] JOIN nom_table1  
ON nom_table_2 = nom_table_1.colonne_1
```

LEFT [OUTER] JOIN :

Vérifie d'abord les informations de la table A, avant de retourner toutes les informations même si elles ne sont pas présentes dans la table 'B'. Les infos non présentes seront retournées en tant que 'NULL'

- On peut préciser les colonnes que l'on souhaite afficher. Elles peuvent être présentes sur une table ou sur l'autre mais il faut préciser son origine
- [OUTER] est implicite et par défaut
- 'ON' sert à indiquer les colonnes à faire correspondre

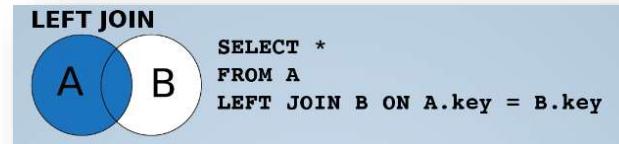
Le risque étant de faire apparaître des champs vides !

Les jointures

LEFT JOIN

La table de 'gauche' est la première table citée, ici « eleves »

```
SELECT
    nom AS Nom,
    prenom AS Prénom,
    date_naissance AS 'Date de naissance',
    classes.niveau AS Classe
FROM
    eleves
        LEFT JOIN
    classes ON classe_id = classes.id
```



Ici, nous avons demandé à afficher la colonne 'classes.niveau' : MySQL a donc affiché exactement ce qui a été demandé.

	Nom	Prénom	Date de naissance	Classe
▶	Bowie	David	1947-01-08	Sixième
	John	Elton	1947-03-25	Sixième
	Cartman	Éric	1989-07-01	segpa
	Norton	Edward	1959-08-18	cinquième
	Nicks	Stevie	1948-05-26	Quatrième
	Kilmister	Lemmy	1945-12-24	Troisième
	Hudson	Saul	1965-07-23	NUL

Slash est bien présent, mais avec une valeur de 'NULL' dans 'Classe'

Les jointures

LEFT JOIN

Première lecture

ID	nom	prenom	date_naissance	classe_id
1	Bowie	David	1947-01-08	6
2	John	Elton	1947-03-25	6
3	Cartman	Éric	1989-07-01	7
4	Norton	Edward	1959-08-18	5
5	Nicks	Stevie	1948-05-26	4
6	Kilmister	Lemmy	1945-12-24	3
7	Hudson	Saul	1965-07-23	8
*	HULL	HULL	HULL	HULL

```
SELECT
  *
FROM
  eleves
  LEFT JOIN
  classes ON classe_id = classes.id
ORDER BY classe_id
```

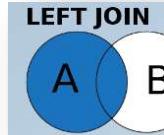
Vérif des correspondances

id	niveau
2	Seconde
3	Troisième
4	Quatrième
5	cinquième
6	Sixième
7	segpa
*	HULL

Table virtuelle

	ID	nom	prenom	date_naissance	classe_id	id	niveau
▶	6	Kilmister	Lemmy	1945-12-24	3	3	Troisième
5	Nicks	Stevie		1948-05-26	4	4	Quatrième
4	Norton	Edward		1959-08-18	5	5	cinquième
1	Bowie	David		1947-01-08	6	6	Sixième
2	John	Elton		1947-03-25	6	6	Sixième
3	Cartman	Éric		1989-07-01	7	7	segpa
7	Hudson	Saul		1965-07-23	8	HULL	HULL

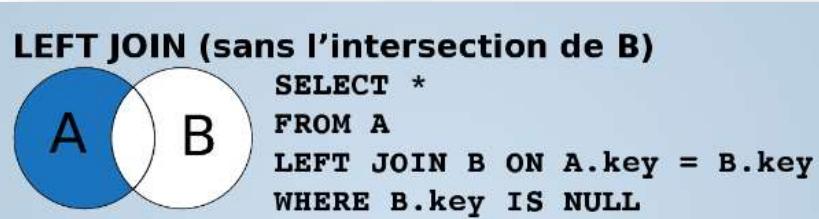
Les informations sont ‘NULL’ car la classe n’existe pas !



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

Les jointures

LEFT JOIN – sans intersection



```
SELECT  
    nom AS Nom,  
    prenom AS Prénom,  
    date_naissance AS 'Date de naissance',  
    classes.niveau AS Classe  
  
FROM  
    eleves  
        LEFT JOIN  
    classes ON classe_id = classes.id  
  
WHERE  
    classes.id IS NULL  
  
ORDER BY classe_id
```

	Nom	Prénom	Date de naissance	Classe
▶	Hudson	Saul	1965-07-23	NULL

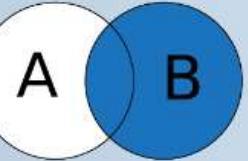
On peut choisir de n'afficher que les données non reprises avec l'argument 'WHERE'

Utile pour vérifier l'affichage et filtrer les données qui ne seraient pas reprises

Les jointures

RIGHT JOIN

RIGHT JOIN



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

```
SELECT * FROM nom_table  
RIGHT [OUTER] JOIN nom_table1  
ON nom_table_2 = nom_table_1.colonne_1
```

RIGHT [OUTER] JOIN :

Vérifie d'abord les informations de la table B, avant de retourner toutes les informations même si elles ne sont pas présentes dans la table 'A'. Les infos non présentes seront retournées en tant que 'NULL'

- On peut préciser les colonnes que l'on souhaite afficher. Elles peuvent être présentes sur une table ou sur l'autre mais il faut préciser son origine
- [OUTER] est implicite et par défaut
- 'ON' sert à indiquer les colonnes à faire correspondre

Le risque étant de faire apparaître des champs vides !

Création d'une base de données sur MySQL Workbench

Les jointures

La table de 'droite' est la deuxième table citée, ici « classes »

```
SELECT
    nom AS Nom,
    prenom AS Prénom,
    date_naissance AS 'Date de naissance',
    classes.niveau AS Classe
FROM
    eleves
        RIGHT JOIN
    classes ON classe_id = classes.id
ORDER BY classe_id
```

Slash n'a pas non plus été repris, ce dernier ne figurant pas dans la première table vérifiée (table de droite)

	Nom	Prénom	Date de naissance	Classe
▶	NULL	NULL	NULL	Seconde
	Kilmister	Lemmy	1945-12-24	Troisième
	Nicks	Stevie	1948-05-26	Quatrième
	Norton	Edward	1959-08-18	cinquième
	John	Elton	1947-03-25	Sixième
	Bowie	David	1947-01-08	Sixième
	Cartman	Eric	1989-07-01	segpa

Aucun élève n'étant en classe de seconde, une ligne a été ajoutée avec toutes les valeurs 'NULL'

Création d'une base de données sur MySQL Workbench

Les jointures

Vérif des correspondances

ID	nom	prenom	date_naissance	classe_id
1	Bowie	David	1947-01-08	6
2	John	Elton	1947-03-25	6
3	Cartman	Éric	1989-07-01	7
4	Norton	Edward	1959-08-18	5
5	Nicks	Stevie	1948-05-26	4
6	Kilmister	Lemmy	1945-12-24	3
7	Hudson	Saul	1965-07-23	8
*	NULL	HULL	HULL	HULL

```
SELECT
*
FROM
    eleves
    RIGHT JOIN
        classes ON classe_id = classes.id
ORDER BY classe_id
```

Table virtuelle

ID	nom	prenom	date_naissance	classe_id	id	niveau
NULL	HULL	NULL	HULL	HULL	2	Seconde
6	Kilmister	Lemmy	1945-12-24	3	3	Troisième
5	Nicks	Stevie	1948-05-26	4	4	Quatrième
4	Norton	Edward	1959-08-18	5	5	cinquième
2	John	Elton	1947-03-25	6	6	Sixième
1	Bowie	David	1947-01-08	6	6	Sixième
3	Cartman	Éric	1989-07-01	7	7	segpa

Bien que vide, notre classe de seconde (ID '2') apparaît

Première vérif

id	niveau
2	Seconde
3	Troisième
4	Quatrième
5	cinquième
6	Sixième
7	segpa
*	NULL

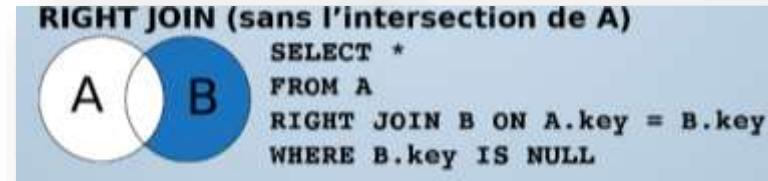
Slash n'est pas présent non plus : la classe ayant l'ID '8' n'existe pas

En sélectionnant toutes les colonnes avec « * », la table virtuelle est un peu plus facile à comprendre

Création d'une base de données sur MySQL Workbench

Les jointures

```
SELECT nom AS Nom,  
       prenom AS Prénom,  
       date_naissance AS 'Date de naissance',  
       classes.niveau AS Classe  
FROM eleves  
      RIGHT JOIN  
       classes ON classe_id = classes.id  
WHERE classe_id IS NULL  
ORDER BY classe_id
```



Result Grid | Filter Rows:

	Nom	Prénom	Date de naissance	Classe
▶	HULL	HULL	NULL	Seconde

On peut choisir de n'afficher que les données non reprises avec l'argument 'WHERE'

Ici on affiche que la classe de seconde, qui ne contient aucun élève

5. Le temps



Le temps

Le temps

Quelques fonctions concernant les dates :

Fonction des jours	Retour
DAY(date)	Jour en nombre entier
DAYOFWEEK(date)	N° du jour (1 à 7)
WEEKDAY(date)	N° du jour (0 à 6)
DAYNAME(date)	Nom du jour
DAYOFYEAR(date)	N° annuel du jour (0 à 366)

Fonctions de semaines	Retour
WEEK(date)	N° de la semaine + DAYOFWEEK
WEEKOFYEAR(date)	N° de la semaine
YEAROFWEEK(date)	Année + N° semaine

Fonctions des mois	Retour
MONTH(date)	N° du mois
MONTHNAME(date)	Nom du mois

Fonctions d'années	Retour
YEAR(date)	Année

Fonctions d'heures	Retour
TIME(datetime)	Extrait TIME de la date
HOUR(x)	Extrait l'heure
MINUTES(x)	Extrait les minutes
SECOND(x)	Extrait les secondes

Fonctionne avec :
- DATETIME
- TIME

Le temps et les fonctions

Le temps

Formater une date dans un string avec DATE_FORMAT() :

```
SELECT DATE_FORMAT(colonne, 'affichage sous forme de string')
```

Exemple :

```
SELECT DATE_FORMAT(NOW(), 'Nous sommes le %d %M de  
l\'année %Y. Il est %l heure %i') AS Date;
```



Date
Nous sommes le 14 November de l'année 2021. Il est 12 heure 21

NOW() renvoie de DATETIME actuel

Le temps

Le temps

Spec.	Description
%d	Jour du mois (00 à 31)
%e	Jour du mois (0 à 31)
%D	Jour du mois + suffixe (1st, 2 nd ...)
%w	N° du jour de la semaine (0 à 6)
%W	Nom du jour (lundi, mardi ...)
%a	Nom du jour en abrégé
%m	Mois (00 à 12)
%c	Mois (0 à 12)
%M	Nom du mois
%b	Nom du mois en abrégé
%y	Année sur 2 chiffres (21 pour 2021)

Spec.	Description
%Y	Année sur 4 chiffres (2021)
%r	Heure complète sur 12h (hh:mm:ss AM/PM)
%T	Heure complète sur 24h (hh:mm:ss)
%h	Heure sur 2 chiffres – 12h (00 à 12)
%H	Heure sur 2 chiffres – 24h (00 à 23)
%I	Heure sur 12h (0 à 12)
%k	Heure sur 24h (0 à 23)
%i	Minutes (00 à 59)
%s ou %S	Secondes (00 à 59)
%p	AM ou PM

Une possibilité pour chaque affichage !

Le temps

Le temps

Calculs d'intervalles entre deux dates :

```
SELECT DATEDIFF('date1', 'date2')  
-- Nombre de jours
```



```
SELECT DATEDIFF(now(), '1945-01-01 15:00:00')
```



Décalage
28076

```
SELECT TIMEDIFF('time1', 'time2')  
-- Nombre de d'heures (TIME)
```



```
SELECT TIMEDIFF(now(), '1945-01-01 15:00:00')
```



Décalage
838:59:59

```
SELECT TIMESTAMPDIFF('unité', 'date1', 'date2')  
-- donne le résultat dans l'unité souhaitée
```



```
SELECT TIMESTAMPDIFF(MONTH, now(), '1945-01-01 15:00:00')
```



Décalage
-922

Fonctionne avec les types DATE ou DATETIME

Le temps

Le temps

Ajout de date avec DATE_ADD() :

```
SELECT DATE_ADD(date, INTERVAL nb unité)
```

Exemple :

```
SELECT DATE_ADD(now(), INTERVAL 3 DAY) AS 'Aujourd\'hui + 3 jours'
```

Fonctionne avec
DATE_SUB pour
soustraire

NOW() renvoie de DATETIME actuel

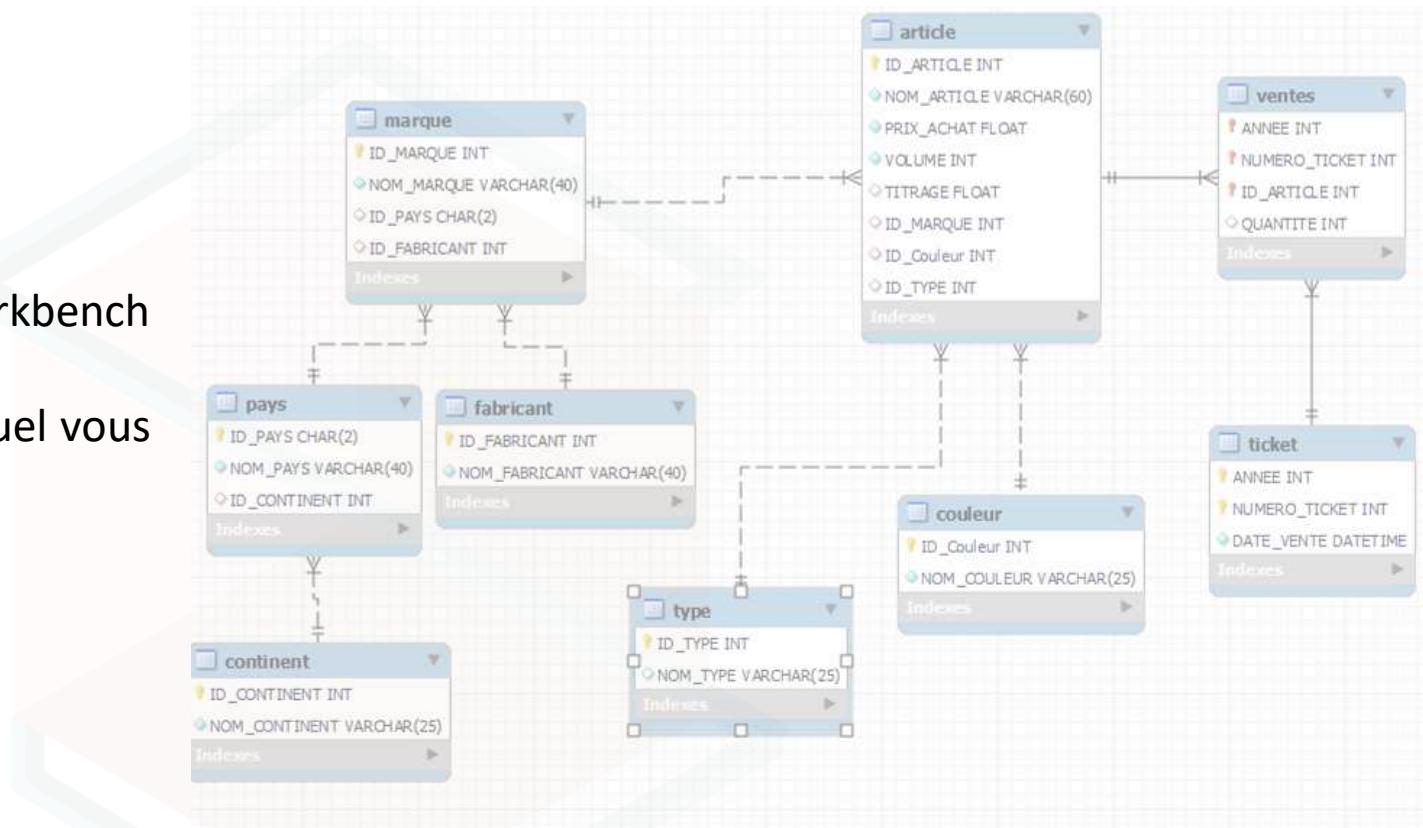
Unités courantes :	Autres unités :
SECOND	MINUTE_SECOND
MINUTE	HOUR_SECOND
HOUR	HOUR_MINUTE
DAY	DAY_MINUTE
WEEK	DAY_MINUTE
MONTH	DAY_HOUR
YEAR	YEAR_MONTH

TRAINING TIME !

TP_BEER

Exercice :

1. Importez le TP Beer dans votre Workbench
2. Créez un document Word dans lequel vous indiquerez pour chaque question :
 1. La commande tapée,
 2. Une capture d'écran du retour
3. Réalisez les exercices 1 à 9



Votre devoir sera à restituer sur l'onglet 'Devoirs' de Teams

6. Les vues



Création d'une base de données sur MySQL Workbench

Les vues

Les vues sont des objets de base de données constitués :

- D'un nom,
- D'une requête de sélection (SELECT)

Objectif :

- Ne pas avoir à retaper systématiquement une (longue) requête SELECT récurrente,
- Peut être utilisé pour réaliser des modifications sur ces requêtes complexes.

En quelque sorte : une vue serait une fonction dont le but unique est l'affichage.

Les vues sont un très bon moyen de gagner du temps !

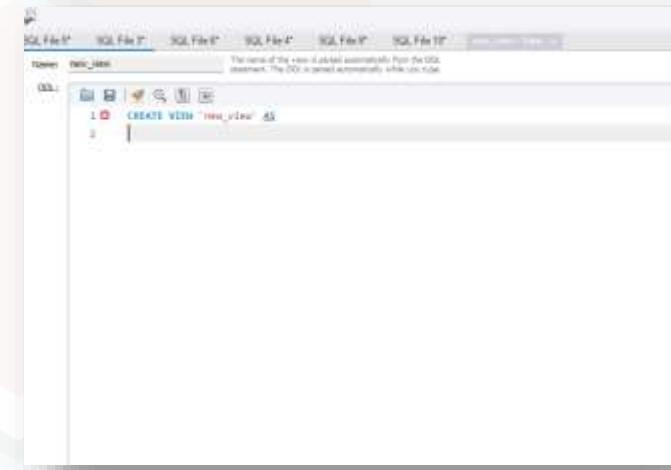
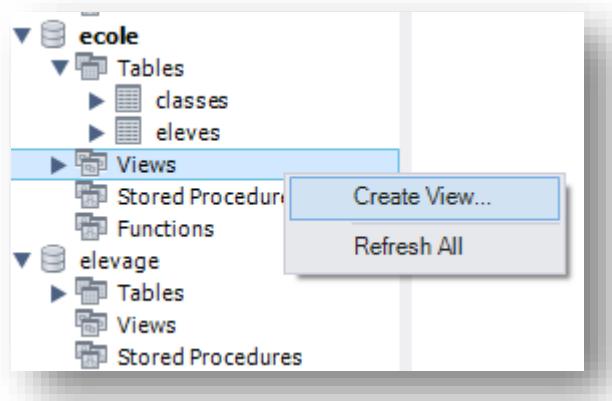
Création d'une base de données sur MySQL Workbench

Les vues

Syntaxe en ligne de commande:

```
CREATE [OR REPLACE] VIEW nom_vue  
AS requete_select;
```

Pré-remplissage via MySQL Workbench :



On peut créer des vue sur des besoins spécifiques pour un utilisateur ou la feature d'une application

Création d'une base de données sur MySQL Workbench

Les vues

La vue ainsi déclarée :

```
CREATE VIEW `Vue_eleves` AS  
SELECT nom AS Nom, prenom AS Prénom, classes.niveau AS Classe  
FROM eleves  
LEFT JOIN classes ON classe_id = classes.id
```

Permet d'appeler la requête SELECT suivante :

```
SELECT * FROM vue_eleves
```

Qui retournera :

	Nom	Prénom	Classe
▶	Bowie	David	Sixième
	Elton	John	Sixième
	Zemmour	Eric	segpa
	Norton	Edward	cinquième
	Nicks	Stevie	Quatrième
	Kilmister	Lemmy	Troisième
	Hudson	Saul	NULL

Mais que va-t-on faire de tout ce temps gagné ? :o

Création d'une base de données sur MySQL Workbench

Les vues

Sur des tables complexes, on peut y ajouter toutes les jointures nécessaires !

Quelques mises en garde :

- Pas de doublons de noms de colonne : utilisez des ALIAS
- On ne peut pas créer une vue contenant une sous-requête SELECT dans le FROM
- La requête ne doit pas faire référence à des variables user ou locales
- Toutes les tables présentes dans la vue doivent exister

Elle peut toutefois contenir :

- Des requêtes incluant 'WHERE'
- Des requêtes incluant 'GROUP BY'
- Des fonctions scalaires, mathématiques ...
- Une autre vue, des jointures, etc...

Les vues sont un très bon moyen de gagner du temps !

Création d'une base de données sur MySQL Workbench

Les vues

Quelques clauses d'algorithme pour la création des vues :

```
CREATE [OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW nom_vue
AS requete_select
```

Les vues sont un très bon moyen de gagner du temps !

Création d'une base de données sur MySQL Workbench

Les vues

MERGE :

Lorsqu'on sélectionne des données, la requête SELECT sera fusionnée avec la clause de sélection :

En plus clair :

Il créera une nouvelle requête (qui correspond à notre vue) à la suite de laquelle il ajoutera notre clause de sélection (WHERE).

Seules les vues de type MERGE pourront être modifiées.

C'est le type de vue le plus performant

Création d'une base de données sur MySQL Workbench

Les vues

TEMPTABLE :

TEMPTABLE pour « Table Temporaire » :

Une table temporaire sera créée afin de permettre l'affichage de nos données.

Une requête WHERE pourra être utilisée pour les filtrer, mais elle sera exécutée après la création de cette table. C'est pourquoi nous ne pourrons pas modifier les données présentes (la clause étant affichée après)

C'est le type de vue le plus performant

Création d'une base de données sur MySQL Workbench

Les vues

UNDEFINED :

C'est l'option par défaut si on ne précise pas d'algorithme.

MySQL choisit lui-même l'option (TEMPTABLE ou MERGE) qu'il va utiliser. Le plus souvent, il appliquera MERGE.

Certains éléments rendront une table incompatible avec MERGE :

- DISTINCT,
- LIMIT,
- Les fonctions d'agrégation (SUM, MAX etc.)
- HAVING et UNION
- Une sous requête dans la clause SELECT

C'est le type de vue le plus performant

Création d'une base de données sur MySQL Workbench

Les vues

Dans notre cas, l'algo qui a été choisi par défaut est visiblement 'TEMPTABLE' :

```
UPDATE Vue_eleves  
SET Prénom = 'Slash'  
WHERE Prénom = 'Saul';
```

Nous utilisons ici les ALIAS que nous avons donné à nos colonnes lors de la création de la vue

Error Code: 1288. The target table Vue_eleves of the UPDATE is not updatable

Nous ne pouvons pas modifier notre table 'eleves' avec cette vue ! Il aurait fallu expliciter l'algo « MERGE »

Création d'une base de données sur MySQL Workbench

Les vues

Il existe beaucoup de restrictions concernant les vues et leur utilité (modifications, insertions...) :

- Les modifications avec UPDATE ne peuvent être faites que si :
 - La modification ne concerne qu'une des deux tables,
 - La clause WHERE ne doit pas contenir de sous requête
- Les vues avec jointure peuvent supporter les requêtes INSERT si
 - Il n'y a QUE des INNER JOIN,
 - L'insertion n'est faite que sur une table

Ceci n'est qu'un échantillon, elles sont très circonstancielles !

7. Les fonctions et agrégations



Les fonctions

Bonus

Calcul directement intégré à la requête :

```
SELECT colonne1, colonne2*X AS alias FROM nom_table;
```

Calcul et modification intégré à la requête :

```
UPDATE nom_table SET colonne1 = colonne1 + X;
```

Quelques manips bonus

Les fonctions

Définition

Définition :

Une fonction est une série d'instruction précises. Elle est définie par son **nom** et ses **paramètres (arguments)**. Quelques exemples de fonctions connues :

- **MIN()** renverra la plus petite valeur
- **PI()** : Renverra le nombre PI avec 5 décimales
- **REPEAT(<argument>, <Nbre X>)** : Répète X fois l'argument

Pas d'espace entre le nom de la fonction et la parenthèse ouvrante !

Les fonctions

Définition

Il existe plusieurs types de fonctions :

- Les fonctions scalaires,
 - S'appliquent à chaque ligne individuellement. Renverra autant d'items que de lignes dans la table

Exemple -ROUND() : Arrondit à l'entier le plus proche

```
SELECT colonne1, colonne2 ROUND(colonne2) FROM nom_table;
```

- Les fonctions d'agrégation
 - Regroupent les lignes en une seule. Renverra donc une seule ligne

Exemple - MIN() : Renvoie la valeur la plus petite

```
SELECT MIN(colonne) FROM nom_table;
```

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Quelques exemples :

`SELECT CURRENT_USER();`

Renvoie l'USER et l'hôte utilisés lors de la connexion

`SELECT USER();`

Renvoie l'USER et l'hôte utilisés lors de l'identification au serveur

`FOUND_ROWS()`

Renvoie le nb de lignes rapporté par la dernière requête

`SELECT CAST('exp' AS TYPE);`

Converti un objet en un autre type (ex : string vers DATE)

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de nombres :

Les arrondis :

```
SELECT CEIL(1.5), CEIL(1.2);
```

Arrondit à l'entier supérieur (ici : 2)

```
SELECT FLOOR(1.5), FLOOR(1.2)
```

Arrondit à l'entier inférieur (ici 1)

```
SELECT ROUND(1.5), ROUND(1.2)
```

Arrondit au plus proche (ici : 2 et 1)

```
SELECT ROUND(1.55, 1), ROUND(1.23, 1)
```

*Arrondit au plus proche, à x décimale près
(ici 1 décimale, renverra 1,6 et 1,2)*

```
SELECT TRUNCATE(1.5, 0), TRUNCATE(1.22, 1)
```

*Tronque de 'x' décimales
(ici 0 et 1 :renverra 1 et 1,2)*

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de nombres :

Les exposants et les racines :

```
SELECT POW(2, 5), POWER(5, 2);  
-- Renverra 32 et 25
```

Multiplie le nombre ‘n’ par lui-même ‘x’ fois

Le nombre ‘n’ peut être un calcul

```
SELECT SQRT(4);
```

Divise le nombre par lui même

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de nombres :

Autres :

`SELECT SIGN(-20), SIGN(23);`

Indiquera en BOOL si le nombre est positif (1) ou négatif (0)

`SELECT FLOOR(1.5), FLOOR(1.2)`

Arrondit à l'entier inférieur (ici 1)

`SELECT ABS(-20)`

Renverra la valeur absolue (sans le ‘-’)

`SELECT MOD(20, 3)`

*Renverra le modulo de 20 divisé par 3
(ici : 2)*

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

Longueur :

```
SELECT BIT_LENGTH('licorne')
```

Indique le nbre de bits d'une chaîne (ici : 56)

```
SELECT CHAR_LENGTH('licorne')
```

Indique le nombre de caractères (ici : 7)

```
SELECT LENGTH('licorne')
```

Retourne le nombre d'octets (ici : 7)

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

Autres :

```
SELECT repeat('licorne', 4)
```

Répète la chaîne 'c' un nombre 'n' de fois

```
SELECT LPAD('licorne', 10, 'x')
```

Complète la chaîne 'licorne' par 'x' dans la limite de '10' caractères.

NB : Tronque la chaîne si '10' est inférieur au nombre de caractères de la chaîne 'licorne'

```
SELECT RPAD('licorne', 10, 'x')
```

Fais la même chose que LPAD mais par la droite

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

TRIM :

Permet de retirer un caractère d'une chaîne.

```
TRIM([[BOTH | LEADING | TRAILING] [caract] FROM] texte);
```

Exemple :

```
SELECT TRIM('    Licorne ') AS both_espace,
       TRIM(LEADING FROM '    Licorne ') AS lead_espace,
       TRIM(TRAILING FROM '    Licorne ') AS trail_espace,
       TRIM('e' FROM 'eeeeLicorneeeee') AS both_e,
       TRIM(LEADING 'e' FROM 'eeeLicorneeee') AS lead_e,
       TRIM(BOTH 'e' FROM 'eeeLicorneeee') AS both_e,
       TRIM('123' FROM '1234ABCD4321') AS both_123;
```



	both_espace	lead_espace	trail_espace	both_e	lead_e	both_e	both_123
▶	Licorne	Licorne	Licorne	Licor	Licorneeee	Licor	4ABCD4321

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

SUBSTRING :

Permet de récupérer une partie d'une chaîne (sous chaîne). Il y a 3 syntaxes possibles :

```
SELECT SUBSTRING('Licorne', 2) AS from2,  
-- Affiche après avoir sauté 1 caractères  
    SUBSTRING('Licorne' FROM 3) AS from3,  
-- Affiche à partir du troisième caractère  
    SUBSTRING('Licorne', 2, 3) AS from2long3,  
-- Affiche 3 caractères à partir du second  
    SUBSTRING('Licorne' FROM 3 FOR 1) AS from3long1;  
-- Affiche 1 caractère à partir du troisième
```



	from2	from3	from2long3	from3long1
▶	icorne	corne	ico	c

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

Recherche :

Rechercher un élément dans une chaîne. Il y a 4 syntaxes possibles :

```
SELECT INSTR('tralala', 'la') AS fct_INSTR,
-- Renverra 4 (index=x 4 de la chaîne)
POSITION('la' IN 'tralala') AS fct_POSITION,
-- Renverra 4 également
LOCATE('la', 'tralala') AS fct_LOCATE,
-- Renverra 4,
LOCATE('la', 'tralala', 5) AS fct_LOCATE2;
-- Renverra 6 (dernière lettre correspond à 6ème index de la chaîne)
```



	fct_INSTR	fct_POSITION	fct_LOCATE	fct_LOCATE2
▶	4	4	4	6

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

Changer la casse :

Changer la casse . Il y a 2 x 2 syntaxes possibles :

```
SELECT LOWER('LiCoRnE') AS minuscule,  
          LCASE('LiCoRnE') AS minuscule2,  
          UPPER('LiCoRnE') AS majuscule,  
          UCASE('LiCoRnE') AS majuscule2;
```



	minuscule	minuscule2	majuscule	majuscule2
▶	licorne	licorne	LICORNE	LICORNE

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

Récupérer X parties d'une chaîne:

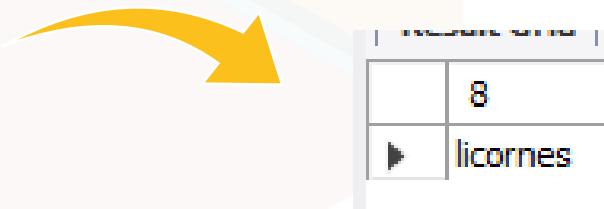
Par la gauche :

```
SELECT LEFT ('Les licornes', 3) AS '3';
```



Par la droite :

```
SELECT RIGHT('Les licornes', 8) AS 8;
```



Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions scalaires – manipulation de chaînes :

Autres :

Inverser une chaîne :

```
SELECT REVERSE('Licornes') AS 'Renversé';
```

Renversé
senrociL

Concaténation :

```
SELECT concat('c1', 'c2')
```

concat('c1', 'c2')
c1c2

```
SELECT concat_ws('X','c1', 'c2')
```

concat_ws('X','c1', 'c2')
c1Xc2

Remplacer une partie de la chaîne

```
SELECT INSERT('Licorne', 3, 4, 'HOHO') AS  
'Remplacement'
```

Remplace 4 caractères à la 3eme position par HOHO
LiHOHOe

Remplacer une partie de la chaîne – autre fonction

```
SELECT REPLACE('Licorne', 'o', '0') AS  
'Remplacement'
```

Remplace les 'o' par '0' dans la chaîne
Lic0rne

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

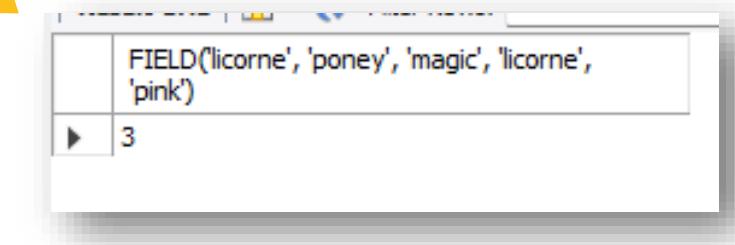
Les fonctions

Fonctions scalaires – manipulation de chaînes :

Récupérer X parties d'une chaîne:

Recherche avec *FIELD*:

```
SELECT FIELD('licorne', 'poney', 'magic', 'licorne', 'pink')
```

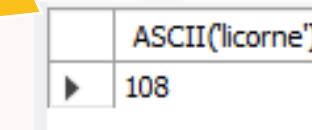


A screenshot of MySQL Workbench showing the results of a query. The query is: `SELECT FIELD('licorne', 'poney', 'magic', 'licorne', 'pink')`. The result set has one row with the value `3`.

	FIELD('licorne', 'poney', 'magic', 'licorne', 'pink')
▶	3

Traduire en ASCII :

```
SELECT ASCII('licorne')
```



A screenshot of MySQL Workbench showing the results of a query. The query is: `SELECT ASCII('licorne')`. The result set has one row with the value `108`.

	ASCII('licorne')
▶	108

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions d'agrégation :

Compter :

Compter tout, sauf les valeurs NULL :

```
SELECT COUNT(*) FROM eleves  
-- Renverra 7
```

Renverra la plus petite valeur

```
SELECT MIN(id) FROM classes
```

Renverra la plus grande valeur

```
SELECT MAX(id) FROM classes
```

Comptera également les valeurs NULL :

```
SELECT COUNT(nom) FROM eleves
```

Renverra somme des éléments

```
SELECT SUM(id) FROM classes
```

Renverra la moyenne des éléments

```
SELECT AVG(id) FROM classes
```

*Ajouter l'attribut
'DISTINCT' permettra
d'éliminer les doublons*

Doc officielle : <https://dev.mysql.com/doc/refman/8.0/en/functions.html>

Les fonctions

Fonctions d'agrégation:

GROUP BY:

Permet de :

- Regrouper les résultats selon un critère
- Elimine les doublons
- Permet l'addition d'items numériques se référant à un même critère

On ne peut sélectionner que deux types d'éléments :

- Ceux qui ont servi dans le GROUP BY
 - Si la colonne n'est pas présente dans les critères, prendra une val random
- Une fonction d'agrégation
 - Pas d'expression courantes, calculs...

Syntaxe sur 1 ou plusieurs critères:

```
SELECT colonne1[, colonne2, ...]  
FROM nom_table  
[WHERE condition]  
GROUP BY colonne1[, colonne2, ...];
```

Exemple :

```
SELECT nom, prenom, classe_id  
FROM eleves  
GROUP BY classe_id
```



	nom	prenom	classe_id
▶	Bowie	David	6
	Zemmour	Eric	7
	Norton	Edward	5
	Nicks	Stevie	4
	Kilmister	Lemmy	3
	Hudson	Saul	8

Doc : <https://sql.sh/cours/group-by>

Les fonctions

Fonctions d'agrégation:

HAVING:

La clause WHERE ne peut pas être utilisée sur les fonctions d'agrégation : HAVING la remplace

NB :

- Having peut être utilisé avec des alias,
- Having peut remplacer WHERE, mais ce n'est pas optimal. A éviter dès que possible donc.

Syntaxe :

```
SELECT colonne1, COUNT(*) FROM table1  
GROUPE BY colonne1  
HAVING COUNT(*) > x
```

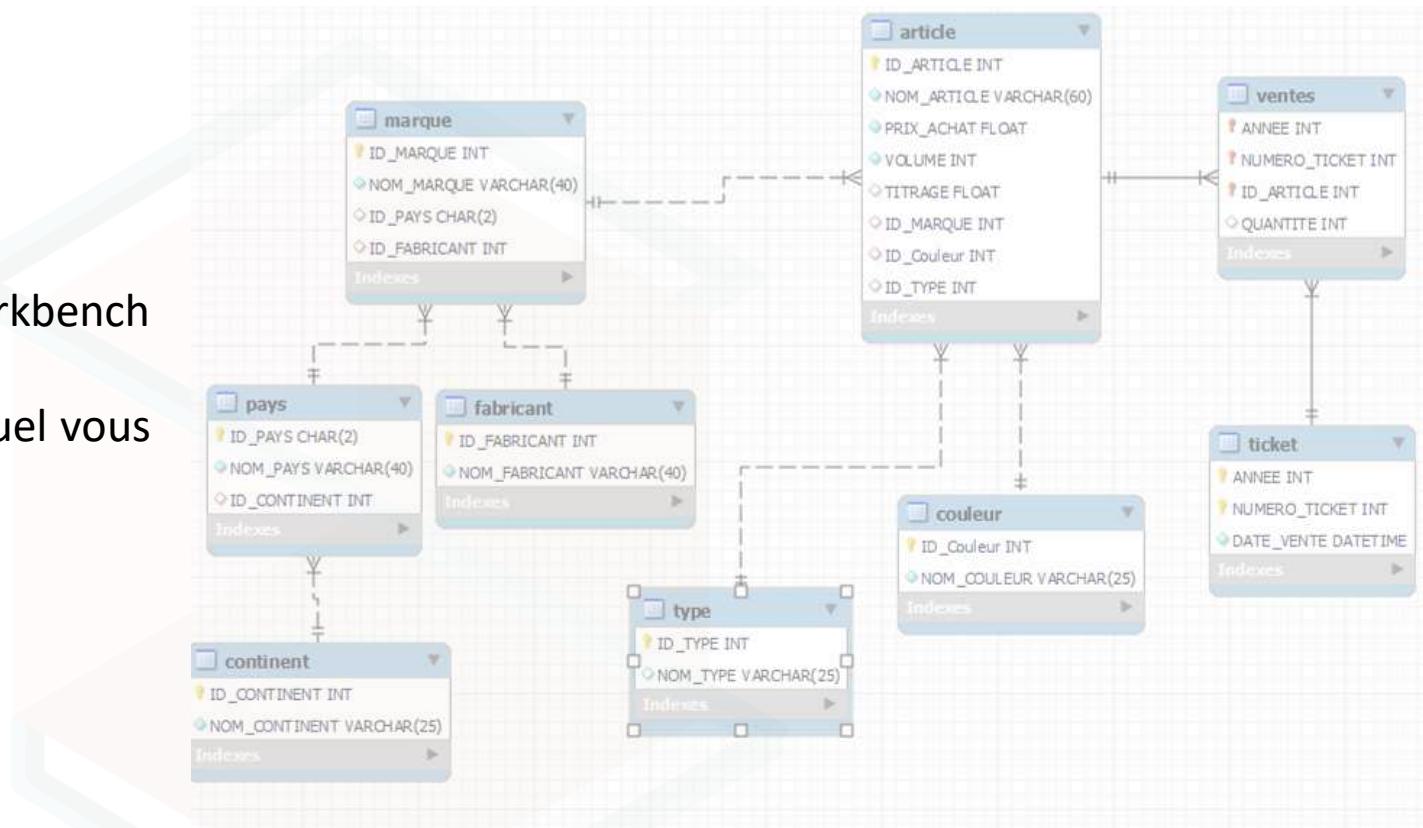
Doc : <https://sql.sh/cours/group-by>

TRAINING TIME !

TP_BEER

Exercice :

1. Importez le TP Beer dans votre Workbench
2. Créez un document Word dans lequel vous indiquerez pour chaque question :
 1. La commande tapée,
 2. Une capture d'écran du retour
3. Réalisez les exercices **10 à 19**



Votre devoir sera à restituer sur l'onglet 'Devoirs' de Teams

8. Les sous requêtes



Les sous-requêtes

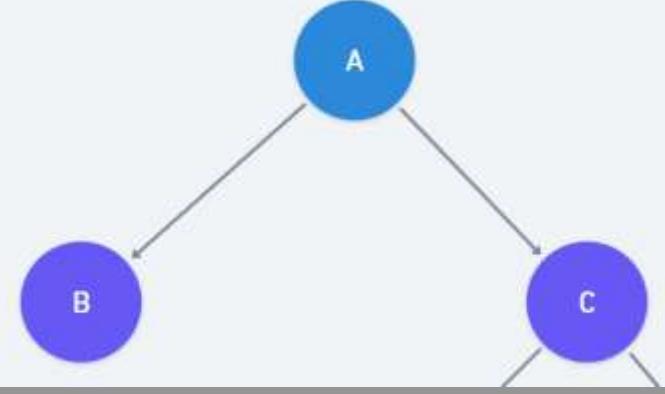
Définition

La « *requete-ception* » (sous requête) :

- Permet d'effectuer une requête à l'intérieur d'une autre requête
- Permet de réaliser plusieurs requêtes en une seule
- Souvent plus clair et intuitif qu'une jointure
 - Mais est moins rapide !
 - Il faut favoriser les jointures quand c'est possible
- La sous requête créé un table intermédiaire que l'on peut utiliser pour formuler notre requête principale

Requête principale

Sous-requêtes



Peut être utilisé avec les requêtes de type :

- *SELECT*
- *INSERT*
- *UPDATE*
- *DELETE*
- *Etc.*

On parle également de « requête imbriquée » ou de « requête en cascade »

Les sous-requête

Syntaxe de sous-requête dans le *FROM*

La sous requête exprimée entre (parenthèses) est exécutée en première

Notre requête finale est ensuite exécutée avec les informations retournées par notre sous requête

```
SELECT * FROM (  
    SELECT * FROM nom_table  
    etc...  
)
```

NB :

- *Il ne peut pas y avoir de doublons d'une requête à l'autre : il faut utiliser des alias*
- *Les rubriques (colonnes) de la requête finale doivent être présente dans la requête intermédiaire et inversement.*

Les sous-requête

Syntaxe de sous-requête dans le *FROM*

```
SELECT nom, prenom, classe_id  
FROM (SELECT nom, prenom, classe_id  
      FROM eleves  
     WHERE date_naissance < '1950-01-01') AS Eleves_6eme_nes_av_1950  
    WHERE classe_id LIKE '6'
```

1 : On recherche les élèves nés avant 1950

	nom	prenom	classe_id
▶	Bowie	David	6
	Elton	John	6

2 : Qui sont en classe de sixième

NB :

- Il ne peut pas y avoir de doublons d'une requête à l'autre : il faut utiliser des alias
- Les rubriques (colonnes) de la requête finale doivent être présente dans la requête intermédiaire et inversement.

Les sous-requête

Syntaxe de sous-requête dans les conditions

Sous requête qui renvoie une valeur :

```
SELECT colonne1, colonne2, colonne3 FROM table1  
WHERE colonne3 =  
    (SELECT colonne1 FROM table2 WHERE nom = 'valeur');
```

-- C'est exactement la même chose qu'avec une jointure

```
SELECT table1.colonne1, table1.colonne2, table1.colonne3  
FROM table1  
INNER JOIN table2 ON table2.colonne1 = table1.colonne3  
WHERE table2.colonne2 = 'valeur';
```

NB :

- *N'oubliez pas qu'il est préférable d'utiliser les jointures si possible !*

Les sous-requête

Syntaxe de sous-requête dans les conditions

```
SELECT
    nom, prenom, classe_id
FROM
    eleves
WHERE
    classe_id = (SELECT
                    id
                FROM
                    classes
                WHERE
                    id = 5)
```

	nom	prenom	classe_id
▶	Norton	Edward	5

```
SELECT
    nom, prenom, classe_id
FROM
    eleves
    INNER JOIN
        classes ON classe_id = classes.id
WHERE
    eleves.classe_id = 5
```

NB :

- N'oubliez pas qu'il est préférable d'utiliser les jointures si possible !

Les sous-requête

Syntaxe de sous-requête dans les conditions

Sous requête qui renvoie une ligne :

```
SELECT * FROM nom_table1
WHERE [ROW](colonne1, colonne2) = (
    SELECT colonneX, colonneY
    FROM table2
    WHERE...);
```

NB :

- *La requête ne fonctionnera pas si la sous requête renvoi plus d'une ligne !*

Les sous-requête

Syntaxe de sous-requête dans les conditions

Conditions IN et NOT IN :

La condition IN() permet de restreindre les résultats à une liste de possibilités

```
SELECT colonne1, colonne2, colonne3 FROM table1  
WHERE colonne3 IN (  
    SELECT colonne1 FROM table2  
    WHERE colonne2 IN ('valeur1','valeur2')  
)
```

```
SELECT nom, prenom, classe_id  
FROM eleves  
WHERE (classe_id) IN (  
    SELECT id FROM classes  
    WHERE niveau IN ('cinquième', 'sixième')  
)
```



	nom	prenom	classe_id
▶	Bowie	David	6
	Elton	John	6
	Norton	Edward	5

Ici, la sous requête va afficher un tableau contenant nos deux valeurs.

La colonne principale utilisera ces deux valeurs comme "filtre" pour sa condition

Les sous-requête

Syntaxe de sous-requête dans les conditions

Sous requête avec ALL :

```
SELECT * FROM table1
-- Selectionne dans la table1...
WHERE colonne1 < ALL (
-- ...les lignes dont la colonne 1 est inférieure à toutes les valeurs de :
    SELECT colonne1 FROM table2
-- Dans la colonne 1 de la table 2...
    WHERE colonne2 IN ('valeur1', 'valeur2')
-- ...les valeurs 1 et 2 sont contenues dans colonne2
);
```

= ANY est équivalent à IN, <> ALL est équivalent à NOT IN
ANY, ALL, et SOME ne fonctionnent qu'avec des sous-requêtes

Les sous-requête

Syntaxe de sous-requête dans les conditions

Exemple :

```
SELECT nom, prenom, classe_id AS niveau  
FROM eleves  
WHERE classe_id < ALL (  
    SELECT id FROM classes  
    WHERE niveau IN ('cinquième', 'sixième')  
)
```



	nom	prenom	niveau
▶	Nicks	Stevie	4
	Kilmister	Lemmy	3

= ANY est équivalent à IN, <> ALL est équivalent à NOT IN
ANY, ALL, et SOME ne fonctionnent qu'avec des sous-requêtes

Les sous-requête

Syntaxe de sous-requête corrélées

```
SELECT colonne1 FROM tableA  
WHERE colonne2 IN (  
    SELECT colonne3  
    FROM tableB  
    WHERE tableB.colonne4 = tableA.colonne5  
);
```

Fait référence à une colonne (ou table) non reprise dans le FROM, mais bien présente ailleurs dans la requête dont elle fait partie :

La sous requête **seule** ne pourra pas être exécutée :

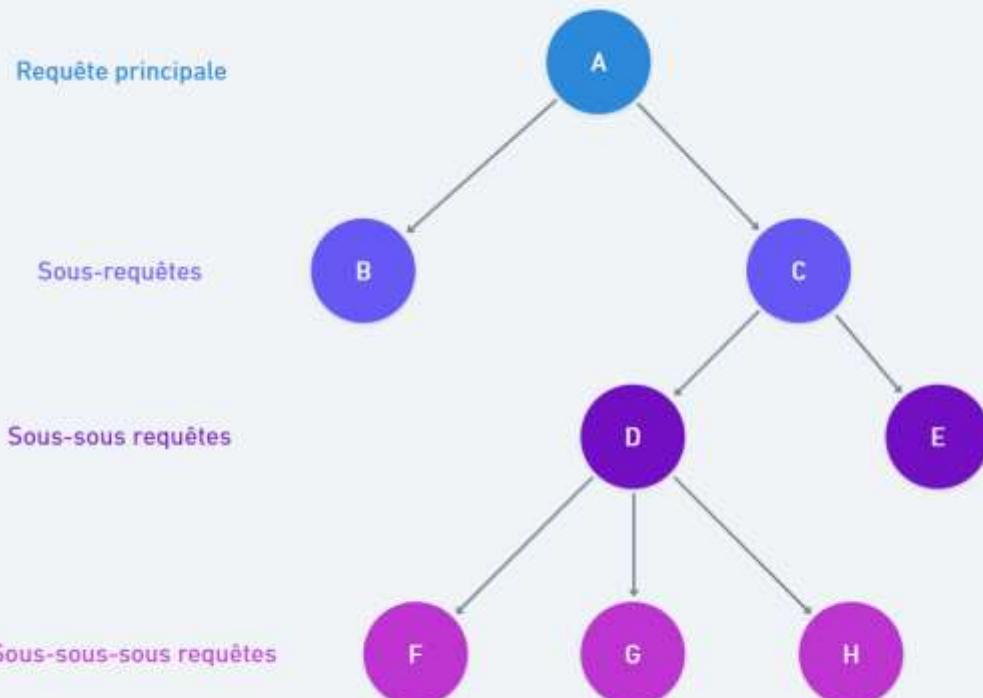
```
SELECT colonne3  
FROM tableB  
WHERE tableB.colonne4 = tableA.colonne5;
```

Seule la table B est sélectionnée dans cette requête, alors qu'on essaie (sans jointure) d'appeler la table A.

= ANY est équivalent à IN, <> ALL est équivalent à NOT IN
ANY, ALL, et SOME ne fonctionnent qu'avec des sous-requêtes

Les sous-requête

Syntaxe de sous-requête corrélées



Les sous requêtes corrélées vont chercher dans les niveau **supérieurs**. Elle ne peut pas chercher dans son propre niveau ou à un niveau inférieur !

Peuvent être corrélées à :

- A : B, C, D, E, F, G et H.
- B : aucune.
- C : D, E, F, G et H.
- D : F, G et H.
- E : aucune.

= ANY est équivalent à IN, <> ALL est équivalent à NOT IN
ANY, ALL, et SOME ne fonctionnent qu'avec des sous-requêtes

Les sous-requête

Syntaxe de sous-requête dans les conditions

Exemple de requête corrélée avec EXISTS :

```
SELECT classes.id, classes.niveau FROM classes  
WHERE EXISTS (  
    SELECT * FROM eleves  
    WHERE classes.niveau = 'Cinquième'  
)
```



	id	niveau
▶	5	Cinquième
●	NULL	NULL

Seule, la sous requête ne peut pourtant pas être utilisée (cette dernière faisant référence à une colonne non présente dans sa table) :

```
SELECT * FROM eleves  
WHERE classes.niveau = 'Cinquième'
```

Error Code: 1054. Unknown column 'classes.niveau' in 'where clause'

Cette requête n'a aucune utilité et est purement illustrative !

Les sous-requête

Syntaxe de sous-requête dans les conditions

La même requête corrélée avec **NOT EXISTS** :

```
SELECT classes.id, classes.niveau FROM classes  
WHERE NOT EXISTS (  
    SELECT * FROM eleves  
    WHERE classes.niveau = 'Cinquième'  
)
```



	id	niveau
▶	2	Seconde
	3	Troisième
	4	Quatrième
	6	Sixième
	7	secpa
	HULL	HULL

Déjà un peu plus « utile » : Notre classe de cinquième n'est plus présente.

Les sous-requête

Syntaxe de sous-requête dans les conditions

Quelques exemples de syntaxe pour modifier des données contenues dans une table à partir d'une autre :

Insertion :

```
INSERT INTO nom_table  
  [(colonne1, colonne2, ...)]  
SELECT [colonne1, colonne2, ...]  
FROM nom_table2  
[WHERE ...]
```

Modification :

```
UPDATE table1  
SET [t1.]colonneX = 'valeur'  
WHERE [t1.]colonneY =  
  (SELECT [t2.]colonneX FROM table2  
  WHERE [t2.]colonneY LIKE '%Valeur2%');
```

NB : On ne peut pas modifier un élément que l'on utilise dans une sous requête !

Les sous-requête

Syntaxe de sous-requête dans les conditions

UNION :

```
SELECT ...
UNION [DISTINCT | ALL]
SELECT ...
```

Définition :

L'union est à intercaler entre deux requêtes SELECT.

Elle permet de créer une table qui reprendra nos deux tables l'une à la suite de l'autre.

On peut chainer des UNION indéfiniment

Elle possède deux options :

- DISTINCT (par défaut si non précisé)
 - Supprimera les doublons
- ALL (conserve les doublons)

Quelques règles :

- Les colonnes doivent être de même type,
 - Sinon, tout sera converti en STRING
- Le SELECT doivent retourner le même nbre de colonnes
- Il faut respecter l'ordre des colonnes
- En cas de mélange entre UNION ALL et UNION DISTINCT
 - Tout sera considéré comme DISTINCT
- Les paramètres LIMIT et ORDER BY s'appliqueront au total des unions

NB : les deux requêtes SELECT doivent retourner LE MÊME NOMBRE DE COLONNES !