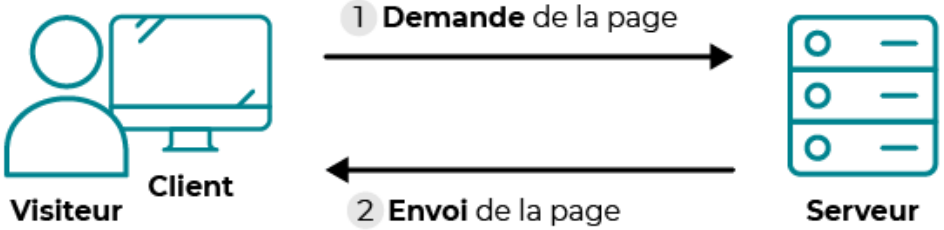
Backend (1) :

Site dynamique (PHP) et gestion de donnée (SQLite/MySQL)

Petit sommaire

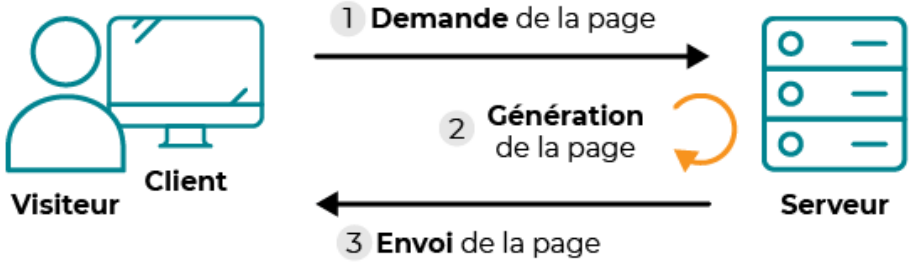
Site statique et site dynamique (avec le procédé client/serveur)

Tout d'abord, un **site statique** est un site « vitrine » pour représenter son entreprise par exemple, réalisé uniquement à l'aide des langages **HTML** et **CSS**. Malheureusement, son contenu **ne peut pas être mis à jour automatiquement** : il faut que le webmaster (programmeur) modifie le code source pour y ajouter des nouveautés. Ce type de site se fait de plus en plus rare aujourd'hui, car dès que l'on rajoute un élément d'interaction (comme un formulaire de contact), on ne parle plus de site statique, mais de site dynamique.



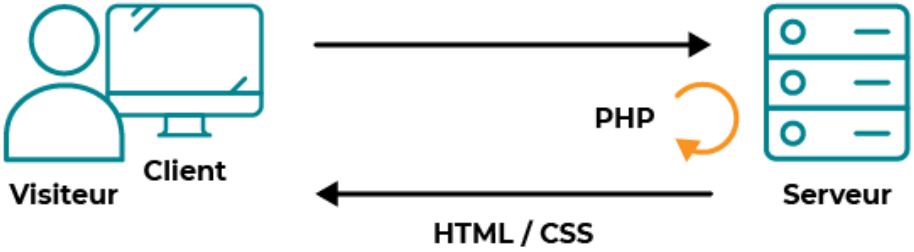
C'est dans ce but, qu'on y trouve des **sites dynamiques** réalisés en plus avec un langage de programmation impératif comme **PHP** (génère la page spéciale pour le visiteur) et d'autres services de données comme **MySQL** (enregistre des données organisées). Il peut **changer le contenu sans l'intervention du webmaster** ! La page web est générée à chaque fois qu'un client la réclame. Ce client peut participer à la vie du site, de poster des messages…C'est précisément ce qui rend les sites dynamiques "vivants" : le contenu d'une même page peut changer d'un instant à l'autre.

ℹ *C'est comme cela que certains sites parviennent à afficher par exemple votre pseudonyme sur toutes les pages. Étant donné que le serveur génère une page à chaque fois qu'on lui en demande une, il peut la personnaliser en fonction des goûts et des préférences du visiteur.*



Récapitulons les différents services et langages qu'on va utiliser :

- **HTML (HyperText Markup Language) :** un langage de balisage qui traduit tout le contenu de la page (déjà vu en formation Udemy)
- **CSS (Cascading Style Sheet) :** une feuille du style qui apporte du design et de la mise en forme sur la page HTML (déjà vu en formation Udemy)
- **PHP (HyperText PreProcessor) :** un langage de programmation qui génère la page Web selon le visiteur
- **MySQL (My Structured Query Language) :** un système de gestion de bases de données de Microsoft en se chargeant du stockage des informations afin de vous aider à les retrouver facilement plus tard comme la liste des membres de votre site, les messages postés sur le forum, etc ; de même le langage qui permet de communiquer avec cette bases de donnée s'appelle SQL



Exécuter en ligne de commande où se trouve le fichier index.php : |php -S localhost:8080|

Les bases du langage de programmation PHP

PHP en code se met tout d'abord sur **des balises** comme HTML n'importe où dans le code (même dans les balises HTML) de la façon |<?php ?>| (ou |<? ?>|, |<% %>|, |<?= ?>|) ou encore « CRTL » + « / » sous VS Code. Très important, avec du PHP, les fichiers avec une extension « .html » seront remplacé par une **extension « .php »**.

La première instruction à connaître est |echo| (affiche une chaîne de caractère) en utilisant |'\n'| pour retourner à la ligne. Pour rappel, une **instruction** commande à l'ordinateur d'effectuer une action précise. On en écrit une par ligne en général, et en PHP elles se terminent par un **point-virgule**. De même, l'instruction echo est suivi par des **guillemets** et parfois en plus des **parenthèses** utilisées pour les instructions précises comme |echo date('d/m/Y h:i:s')|, comme |<?php echo 'ceci est un texte' ; ?>| ou |<?php echo ('ceci est un autre texte') ; ?>|. En parallèle au python, |echo| et |print| sont des instructions identiques. Il faut savoir qu'on a aussi le droit de demander d'afficher des balises. Par exemple, le code suivant fonctionne : |<?php echo "Ceci est du texte "; ?>|. Le mot « texte » sera affiché en gras grâce à la présence des balises et . Puis, aussi pour afficher des guillemets, il faut tout simplement ajouter **un slash avant le guillement** (| \' |). Info en plus, pour être sûr de mettre à jour le fuseau horaire, il faut tout simplement ajouter au début du code en php |<?php date_default_timezone_set('Europe/Paris'); ?>|.

- *Les variables dans un champs de caractères à double guillemets retransmettront leur valeur directement en texte, à l'inverse du simple guillemet qui donnera juste le nom de la variable.*

Pour continuer les débuts, on y trouve l'éléments indispensables pour être repérer personnellement et par une équipe dans notre code : les **commentaires**. On y trouve deux type de commentaires, les **monolignes** | // | et les **multilignes** | /*…*/ | comme CSS.

De même, pour le terminale, il ne faut pas oublier d'insérer le dossier du exécutable « php.exe » dans les variables environnements du système Windows. Puis il faut modifier le fichier « settings.json » du VS code en insérant (entre guillemets) le fichier exécutable « php.exe » en ajoutant avant « "php.validate.executablePath": ».

Voir les erreurs PHP

PHP affiche une page blanche (une page de navigateur sans contenu) quand le script plante pour des raisons de sécurité. "*Moins l'utilisateur en sait sur mon application, mieux mon application se portera.*". De même autre information utile, PHP est **configurable** ! Elle se fait sous un fichier « php.ini ». Pour le savoir, il faut mettre l'instruction PHP | phpinfo(); | et réactualiser la page

du serveur. Dans cette nouvelle page d'information, on obtient la version PHP (moi, la 7.4.1), le type de server web (moi, Apache) et la localisation du (ou des fichiers) de configuration pour PHP (ce qui est important). D'après ce chemin d'accès, vous pouvez ouvrir « php.ini » et modifier deux éléments sur des lignes qui n'ont pas de commentaires (point-virgule en début de ligne de code) :

- ✓ La clé de configuration | error_reporting | a la valeur | E_ALL | (toutes les erreurs seront transmises) ;
- ✓ La clé de configuration | display_errors | a la valeur | On | (les erreurs seront affichées).

Enfin, il faut juste simplement redémarrer le serveur. Et voilà, quand une page PHP aura une erreur, on saura la ligne du code où se trouve cet intrus connu par tous les programmeurs. Cette méthode a un inconvénient car elle n'assure pas la sécurité de votre page. Les visiteurs sauront où se trouve le fichier et la ligne d'erreur.

Programmer en orienté objet en PHP (version simple)

Un code PHP **structuré autour d'objet** qui la compose est une manière de cibler les données selon des catégories, des éléments ou encore des personnes où on va leur(s) attribuer des variables. Ceux qu'on va étudier s'appelleront des **objets « métiers »** comme :

- ❖ Objet Utilisateur : Nom, Email, Mot de passe, Âge
- ❖ Objet Recettes : Titre, Corps, Statut d'activation

Les variables en PHP

Une variable est une **information stockée en mémoire temporairement** dans l'objectif est de **pouvoir varier**. Sauf que les variables existe que quand la page PHP est en train de se générer (puis ils sont supprimés). Ce n'est donc pas un fichier qui reste stocké sur le disque dur, mais une petite information temporaire présente en mémoire vive. Comme chaque langage de programmation, elle a un **nom** (qui permet de la reconnaître) et une **valeur** (l'information qu'elle contient). Comme python et C++, il y a différents type de variable (***string*** *: un texte avec des guillemets,* ***int*** *: un nombre entier,* ***float*** *: une valeur décimal et* ***bool*** *: true/false*) avec en plus le type « rien » (| NULL |), une **absence** de type. Quand vous nommez des variables, évitez les accents, les cédilles et tout autre symbole : PHP ne les apprécie pas trop…

Une variable s'écrit en PHP :

- ❖ D'abord, on écrit le symbole "dollar" ($) : il précède toujours le nom d'une variable. C'est comme un signe de reconnaissance, si vous préférez : ça permet de dire à PHP "J'utilise une variable" ;
- ❖ Ensuite, il y a le signe "égal" (=) ;
- ❖ À la suite, on ajoute la valeur de la variable (*où on peut utiliser des additions, des soustraction, des multiplications, des divisions et des modulos*) ;
- ❖ Enfin, il y a l'incontournable point-virgule (;) qui permet de terminer l'instruction.

Pour les faire afficher, il n'y a plus qu'à les utiliser dans des echo sans parenthèse et sans guillemets, sauf quand on veut inclure du texte (avec des guillemets simples). Le texte et les variables seront **séparés par un point** tel que | 'J'ai ' . $age . ' ans' |.

🛈 *Dans les langages de programmation, on utilise souvent la convention* ***camelCase*** *pour les noms (cela fait référence aux bosses d'un chameau) où on utilise une majuscule pour "détacher" visuellement les mots et les rendre plus lisibles (éviter de faire des espaces).*

🛈 *Un Modulo % est le reste d'une division (euclidienne). Par exemple, 5/2 est égale à 2 avec 1 en reste (%). Il permet aussi de savoir dans une condition, si le premier nombre est divisible par le second.*

Les conditions en PHP

L'une des fondamentales avec les sites dynamiques est la condition. Pour commencer, on a « **if…else** » utilisés pour les conditions simples et courtes, intégrées avec des accolades. Pour cela, je vais les comparer à Python. On a la même forme et les mêmes symboles : | ==, >, <, >=, <=, !== | sauf à quelques exceptions :

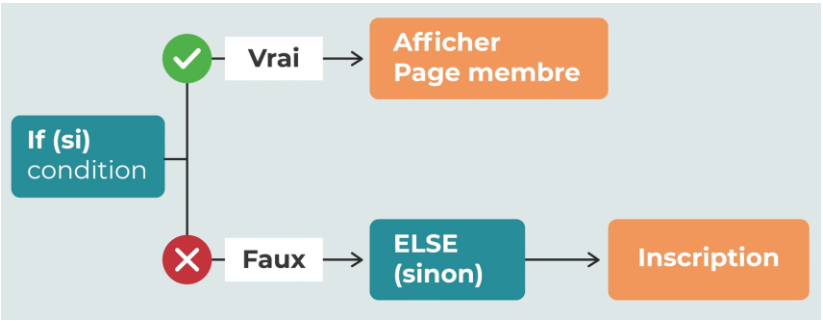
- ✓ | elseif | au lieu de « elif »
- ✓ | && | au lieu de « and »
- ✓ | || | au lieu de « or »
- ✓ | === | permet de vérifier si les deux entrés sont les mêmes au niveau de la valeur et du type.

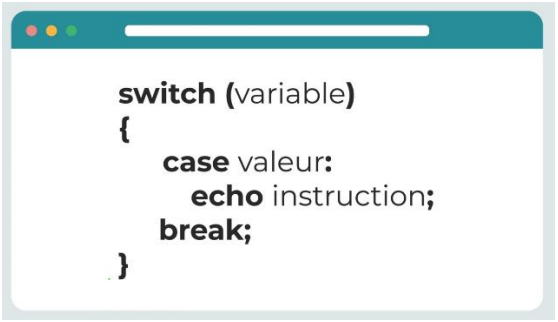
Exemple :

```php
<?php
$isEnabled = true;
$isOwner = false;

if ($isEnabled && $isOwner) {
    echo 'Accès à la recette validé ✅';
} else {
    echo 'Accès à la recette interdit ! ❌';
} ?>
```

Pour les conditions booléens, on inclut juste la variable et on ajoute avant | ! | pour « Faux ». De même, on peut faire aussi du ligne par ligne sur le code HTML tel que ce code (en ajoutant à la fin un | endif |) :

```php
<?php $chickenRecipesEnabled = true; ?>
<?php if ($chickenRecipesEnabled): ?> <!-- Ne pas oublier le ":" -->
<h1>Liste des recettes à base de poulet</h1>
<?php endif; ?><!-- Ni le ";" après le endif -->
```





Pour une série de conditions à analyser, on utilise le **switch** pour rendre le code plus clair. Il se base sur plusieurs conditions d'une même variable (en utilisant des | case | pour analyser chaque cas) qui a la particularité de tester qu'avec l'égalité (*aucun d'autres symboles ne peut être utilisé*) avec beaucoup moins d'accolades. Par rapport à | elseif |, PHP exécute toutes les instructions | case | qui suivent se finissant chacune par | break |. La valeur de base se nomme | default | (*comparable à un else*).

Rarement, il existe une autre façon d'exprimer une condition avec les ternaires. Il a pour but de tester la valeur d'une variable dans une condition et affecter une valeur à une variable selon que la condition est vraie ou non. Par exemple, pour comparatif, c'est une condition condensée d'un if/else sur résultats de valeur booléenne :

```php
<?php
$userAge = 24;

$isAdult = ($userAge >= 18) ? true : false; /* Si la personne a plus de 17 ans, le résultat de la variable $userAge est "true" et l'inverse est "false" */

// Ou mieux, dans ce cas précis
$isAdult = ($userAge >= 18);
?>
```

Les tableaux PHP, la variable sous la forme d'une liste

Un **array** est ce qu'on appelle une variable « **tableau** » comparable aux listes sur Python. Elle n'a pas qu'une valeur, mais une liste de valeur (vous pouvez d'ailleurs en mettre autant que vous voulez). Dans un tableau, les valeurs sont rangées dans des « **cases** » différentes, qui peuvent être identifiées par des **clés**, mises entre crochets. Pour créer un tableau en PHP, on liste ses valeurs entre crochets []. De surcroît, on est obligé d'utiliser des crochets pour indiquer dans quelle « case » on doit aller chercher l'information (par exemple, pour afficher une(des) valeur(s) d'un tableau avec | echo | => | echo $variable[5] |) ; sinon PHP ne sait pas quoi récupérer. ❶ *Comme sur Python, un tableau numéroté commence toujours à la case n° 0 !*

Pour être plus précis, il existe 2 types de tableau :

- Les **tableaux numérotés** permettent de stocker une série d'éléments du même type, comme des prénoms. Chaque élément du tableau contiendra alors un prénom.

```php
<?php $users = ['Socca', 'Couscous', 'Sushi', 'Salade César',];

// La fonction array permet aussi de créer un array
$usersTest = array('Socca', 'Couscous', 'Sushi');
$recipes[] = 'Socca'; // Créera $recipes[0]
$recipes[] = 'Couscous'; // Créera $recipes[1]
$recipes[] = 'Sushi'; // Créera $recipes[2]

//On peut insérer des valeurs d'une autre façon de même
$recipesTest[0] = 'Socca';
$recipesTest[1] = 'Couscous';
$recipesTest[2] = 'Sushi';

echo $users; // Cela affichera : Couscous
?>
```

- Les **tableaux associatifs** permettent de découper une donnée en plusieurs sous-éléments. Par exemple, une adresse peut être découpée en nom, prénom, nom de rue, ville… ❶ *La première valeur numérotée, ajoutée à ce tableau aura l'index 0.*

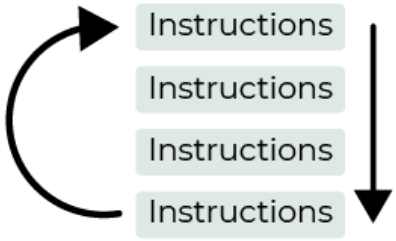
```php
<?php $adresse = [
    'firstName' => 'Mathuww',
    'surname' => 'Hiha',
    'addressName' => '249_chemin_du_vallonnelle_d\'Osgard',
    'postCode' => '06254',
    'town' => 'Moginelle'
];

//On peut insérer des valeurs d'une autre façon de même
$adresse['surname']='Darnoguilem';

echo $adresse['firstName'] .'<br/>' . $adresse['surname'] .'<br/>' . $adresse['addressName'] .'<br/>' . $adresse['postCode'] .'<br/>' . $adresse['town']; // Afficher toutes les valeurs de la variable $adresse
?>
```

Les boucles en PHP

Une boucle est une forme de condition dans un but de répétition. On peut y retrouver une condition et tant qu'elle n'est pas acquise, les instructions se répètent. Attention, il faut vérifier que la condition soit fausse une fois pour éviter le cas de figure d'une boucle qui s'exécutera à l'infini (*comme montre l'image ci-dessous*). C'est toujours comparable au Python comme on le voit avec la première type de boucle « **while** » (tant que…) à utiliser de préférence lorsqu'on ne sait pas par avance combien de fois la boucle doit être répétée. Elle est simple et plus flexible, sans oublie de certains paramètres comme l'incrémentation de la variable. On peut toujours sortir de la boucle avec un | break |.

ℹ *Info dans le cas de figure d'une boucle infinie, PHP refuse normalement de travailler plus d'une quinzaine de secondes. Il s'arrêtera tout seul s'il voit que son travail dure trop longtemps, et affichera un message d'erreur.*

```php
<?php
$lines = 1;

while ($lines <= 100) {
    echo 'Je dois être concentré quand j\'apprends
pendant j\'éxécute ' . $lines . ' lignes de code.' .'<br />';
    $lines++; // $lines = $lines + 1
}
?>
```



Pour compléter les boucles, on a aussi le type « **for** » qu'on utilise lorsqu'on veut répéter des instructions sur un nombre précis de fois. C'est semblable au « while » sauf qu'on y trouve des paramètres de précisions :

- ✓ Le premier sert à l'**initialisation** qui est la valeur que l'on donne au départ à la variable ;
- ✓ Le second, c'est la **condition**, comme pour le « while » : tant que la condition est remplie, la boucle est réexécutée ; à l'inverse, dès que la condition ne l'est plus, on en sort ;
- ✓ Enfin le troisième, c'est l'**incrémentation** qui permet de choisir le nombre de répétition, de saut d'étape de la boucle (« de combien en combien, j'exécute les valeurs de la variable » : de 1 en 1 => $test++ ou de 2 en 2 => $test=$test+2)

```php
<?php $mathuww = ['Mathuww Darnoguilem', 'mathuww@darnoguilem.com', '123Mat,!', 15];
$mathieu = ['Mathieu Nebra', 'mathieu.nebra@exemple.com', 'devine', 33];
$nora = ['Nora Camille', 'nora.camille@exemple.com', 'the@tr€2022', 18];
$users = [$mathuww, $mathieu, $nora];

for ($lines = 0; $lines <= 2; $lines++) //La boucle commence à l'index 0 avec une itération de 1 en 1
($lines++)
{
   echo $users[$lines][0] . ' a l\'adresse e-mail : ' . $users[$lines][1] . '<br />';
} ?>
```

Autre exemple intégré avec du HTML sur les boucles :

```php
<?php
// Déclaration du tableau des recettes
$recipes = [
    ['Socca','[...]','mathuww@darnoguilem.com',true,],
    ['Sushi','[...]','mathuww@darnoguilem.com',false,],
];
$Hello=0; ?>

<!DOCTYPE html>
<html>
<head>
    <title>Affichage des recettes</title>
</head>
<body>
    <ul>
        <?php for ($lines = 0; $lines <= 1; $lines++): ?>
            <li><?php echo $recipes[$lines][0] . ' (' . $recipes[$lines][2] . ')'; ?></li>
        <?php endfor; ?>
        <?php while($Hello<3): ?>
            <li><?php $Hello++; echo $Hello . ' coucou';?></li>
        <?php endwhile; ?>
    </ul>
</body>
</html>
```

Pour approfondir, il existe une boucle « for » spécialisé pour les tableaux : le « **foreach** » qui passe en revue chaque ligne du tableau. La boucle s'arrête lorsqu'on a parcouru tous les éléments de l'array. Lors de chaque passage, elle met la valeur de cette ligne dans une variable temporaire (par exemple $test). Dans la parenthèse, on met d'abord le nom du tableau, ensuite le mot clé | as | (qui signifie « en tant que »), puis le nom le nom d'une variable que vous choisissez, et qui va contenir tour à tour chacune des valeurs du tableau. On peut aussi récupérer la clé (≠valeur) de la variable formulée comme ceci | <?php foreach($recipe as $test => $testValue) ?> | avec | $test | (variable temporaire attribuée aux clés du tableau) et | $testValue | (variable temporaire attribuée aux valeurs du tableau).

ℹ *Entre les accolades, on n'utilise donc que la variable temporaire qui a un nom unique dans le code.*

```php
<?php $recipes = [
    [
        'title' => 'Socca',
        'recipe' => '',
        'author' => 'mathuww@darnoguilem.com',
        'is_enabled' => true,
    ],
    [
        'title' => 'Sushi',
        'recipe' => '',
        'author' => 'nora.camille@exemple.com',
        'is_enabled' => true,
    ],
];

foreach($recipes as $recipe) {
    echo $recipe['title'] . ' contribué(e) par : ' . $recipe['author'] . '<br/>';
} ?>
```

Autre exemple « foreach » avec les variables temporaires différenciées aux valeurs et aux clés du tableau :

```php
<?php $recipes = [
    'title' => 'Sushi',
    'recipe' => 'Etape 1 : préparer les feuilles d\'algues ; Etape 2 : euh ...',
    'author' => 'nora.camille@exemple.com',
    'is_enabled' => true,
];

foreach ($recipes as $test => $testValue)
{
    echo '[' . $test . '] vaut ' . $testValue . '<br/>';
} ?>
```

Pour compléter les boucles, on a des commandes permettant de faire une certaine recherche dans des tableaux PHP. En premier, | array_key_exists | vérifie si une clé existe dans le tableau (*la fonction renvoie un booléen*). En deuxième, | in_array | vérifie si une valeur existe dans le tableau (*la fonction renvoie un booléen*). Pour finir, | array_search | récupère la clé d'une valeur dans le tableau (*la fonction renvoie la clé correspondante si elle a trouvé la valeur, sinon elle renvoie false*). Ces commandes se constituent toujours de la commande avec la recherche et la variable séparées d'une virgule et entourées de parenthèses.

```php
<?php $recipes = [
[
        'title' => 'Socca',
        'recipe' => 'Etape 1 : prendre un contenant en y ajoutant 250g de farine, 3 cuillères à soupe
        d\'huile d\'olive, du sel et du poivre ; Etape 2 : mélanger en y incorporant au fur à mesure 50cl
        d\'eau ; Etape 3 : attendre minimum 1h ; Etape 4 : cuire la socca
        selon vos outils de cuissons (four à pizza, four classique, poële, ...)',
        'author' => 'mathuww@darnoguilem.com',
        'is_enabled' => true,
    ],
    [
        'title' => 'Sushi',
        'recipe' => 'Etape 1 : préparer les feuilles d\'algues ; Etape 2 : ...',
        'author' => 'nora.camille@exemple.com',
        'is_enabled' => true,
    ],
    [
        'title' => 'Salade César',
        'recipe' => 'En cours',
        'author' => 'mathuww@darnoguilem.com',
        'is_enabled' => false,
    ],
]; ?>
<!DOCTYPE html>
<html>
<head>
    <title>Affichage des recettes</title>
    <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
        rel="stylesheet">
</head>
<body>
    <div class="container">
        <h1>Affichage des recettes</h1>
        <!-- Boucle sur les recettes -->
        <?php foreach($recipes as $recipe) : ?>
            <!-- si la clé existe et a pour valeur "vrai", on affiche -->
            <?php if (array_key_exists('is_enabled', $recipe) && $recipe['is_enabled'] == true): ?>
```

```
                <article>
                    <h3><?php echo $recipe['title']; ?></h3>
                    <div><?php echo $recipe['recipe']; ?></div>
                    <i><?php echo $recipe['author']; ?></i>
                </article>
            <?php endif; ?>
        <?php endforeach ?>
    </div>
</body>
</html>
```

Afficher le code d'un tableau PHP

Il y a une fonctionnalité intéressante pour débuguer son code, celui d'afficher rapidement son code PHP sur un navigateur Web avec « print_r ». C'est une sorte de « echo » spécialisé dans les tableaux. Par contre, il a une spécialité de ne pas renvoyer du code HTML comme |
 | pour les retours à la ligne. Pour bien les voir, il faut donc utiliser la balise HTML | <pre> | et | <pre/> | qui nous permet d'avoir un affichage plus correct. Au fait, on n'affiche jamais des choses comme cela aux visiteurs.

```
<?php $recipes = [
    [
        'title' => 'Sushi',
        'recipe' => 'Etape 1 : préparer les feuilles d\'algues ; Etape 2 : euh ...',
        'author' => 'nora.camille@exemple.com',
        'is_enabled' => true,
    ],
    [
        'title' => 'Salade César',
        'recipe' => 'En cours',
        'author' => 'mathuww@darnoguilem.com',
        'is_enabled' => false,
    ],
];

echo '<pre>';
print_r($recipes);
echo '</pre>'; ?>
```

Qu'est-ce que le SQL (langage sur les bases de données relationnelles) ?

SQL, pour **S**tructured **Q**uery **L**anguage, est un langage qui permet d'interroger une base de donnée relationnelle afin de pouvoir modifier ou récupérer des informations. **Les bases de données relationnelles** permettent de sauvegarder les informations sous forme de tableau à **2 dimensions** (table => feuille de calcul /relation) à travers des colonnes et des lignes. Il est en lien avec des gestions de bases de données un peu différentes comme SQLite ou encore MySQL.

Ces bases de données vont s'avérer utiles pour sauvegarder les informations qui permettront plus tard de générer des pages dynamiquement. Par exemple dans le cadre d'un site de recette de cuisine, on ne peut pas se permettre de créer autant de pages HTML que l'on a de recettes. Grâce à une base de données, on pourra sauvegarder les informations, utiliser le langage SQL pour les récupérer et pour générer les différentes pages automatiquement.

L' installation VS Code de SQLite (et la première requête : « CREATE TABLE »)

Tout d'abord, pour configurer le système de gestion de base de données (SGBD) SQLite pour l'utiliser au sein de l'éditeur Visual Studio Code, il faut installer l'extension SQLite. D'après un fichier SQLite, on peut tester cette base de données en faisant un clic droit et en sélectionnant l'option « Open Database » ce qui déploiera un nouveau panneau en bas à gauche de notre explorateur de fichier.

On pourra alors depuis ce panneau créer un nouveau fichier (dans le « SQLITE EXPLORER » avec « New Query ») qui nous servira à taper nos commandes (comme | SELECT 1 |) et à les exécuter avec « Run Query » (ou encore « Ctrl + Shift + Q »).

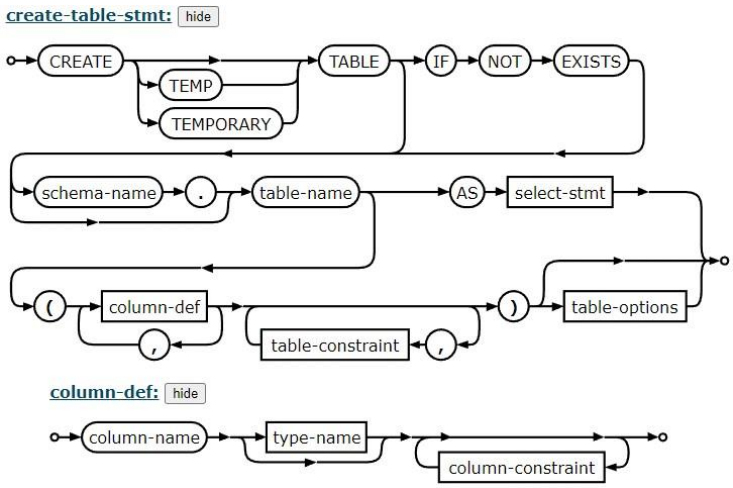
- *Un fichier SQLite a pour extension « .sqlite ».*
- *De même, on peut le faire sous l'application « sqlite » officiel avec le terminale mais aussi sous « TablePlus » avec l'onglet « Open SQL editor Ctrl + E » => l'avantage de ce logiciel est d'exécuter plusieurs requêtes en même temps avec « Run All ».*
- *Les mots-clés sont en majuscule à cause du convention SQL (mais on peut toujours les écrire en minuscules).*

Puis, nous allons analyser/utiliser la requête SQL « CREATE TABLE » qui permet de créer une table et de définir l'ensemble des colonnes qui la compose sous la forme | CREATE TABLE nom-table (nom-colonne et type-valeurs-colonne, ...) ; |. Et SQLite, pour les types de valeurs, elle en a beaucoup assez basique et parfois inutile comme le révèle le tableau :

Exemple de datatype de SQLite	*Le type de résultat des valeurs*
1. **INT** → 4 octets : nombre entier entre -2147483648 à 2147483647	**INTENGER** (*nombre entier*)
2. **INTENGER** → synonyme de « INT »	
3. **TINYINT** → 1 octet : nombre entier entre -128 à 127	
4. **SMALLINT** → 2 octets : nombre entier entre -32768 à -32767	
5. **MEDIUMINT** → 3 octets : nombre entier entre -8388608 à -8388607	
6. **BIGINT** → 8 octets : nombre entre -9223372036854775808 à -9223372036854775807	

7. **VARCHAR (N)** → **N** est le nombre de caractère choisi (de 0 à 255)	**TEXT** (*champ de texte*)
8. **TEXT**	
9. **BLOB** (collection de données binaires stockées en tant qu'entité unique dans un système de gestion de bases de données : images, audios, multimédias, code binaire) → Sur MySQL, le maximum d'octets est 65 535 octets	**BINAIRE** (*je crois, ba, c'est ce type sur MySQL*)
10. **FLOAT** → 4 octets de nombres décimaux = valeur approchée	**REEL** (*nombre à virgule*)
11. **DOUBLE** ou **DOUBLE PRECISION** → 8 octets de nombre décimaux = valeur approximative	
12. **DECIMAL(P, D)** ou **NUMERIC(P, D)** → = valeur exacte → **P** est la précision qui représente le nombre total de chiffres significatifs (de 1 à 65) → **D** est l'échelle qui représente le nombre de chiffres après la virgule (de 0 à 30)	**NUMERIC** (*conversion en REEL ou INTENGER*)
13. **BOOLEAN** → **true ou false** (1 ou 0)	
14. **DATE**	
15. **DATETIME**	

- *Plus d'information sur « CREATE TABLE » : https://www.sqlite.org/lang_createtable.html et sur ses datatypes : https://www.sqlite.org/datatype3.html.*



- *Nous pouvons toujours voir l'évolution de la table avec ses colonnes sous VS code avec le « SQLITE EXPLORER ».*

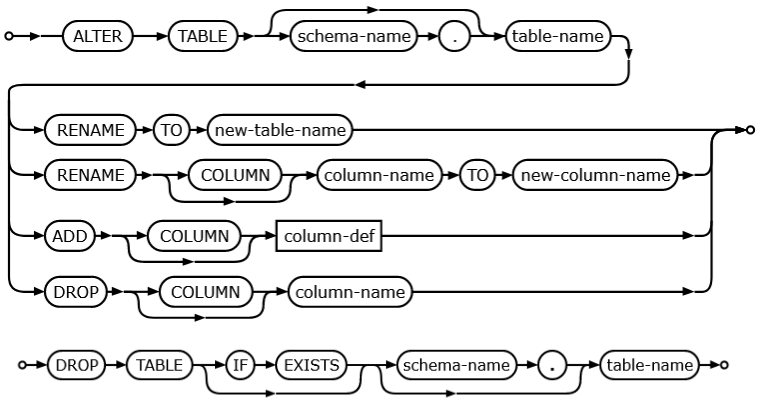
Un exemple de « CREATE TABLE » :

```sql
-- SQLite
CREATE TABLE recipes (
    title VARCHAR(70),
    content TEXT,
    slug TEXT,
    duration SMALLINT,
    online BOOLEAN,
    create_at DATETIME,
    email TEXT,
    author VARCHAR(18)
);
```

- *Sur **les fichiers SQL** (les fichiers où on exécute des requêtes), on peut faire des commentaires monolignes avec | -- | au début de chaque ligne ou de même, on a la possibilité d'utiliser des commentaires multilignes avec | /* … */ |.*

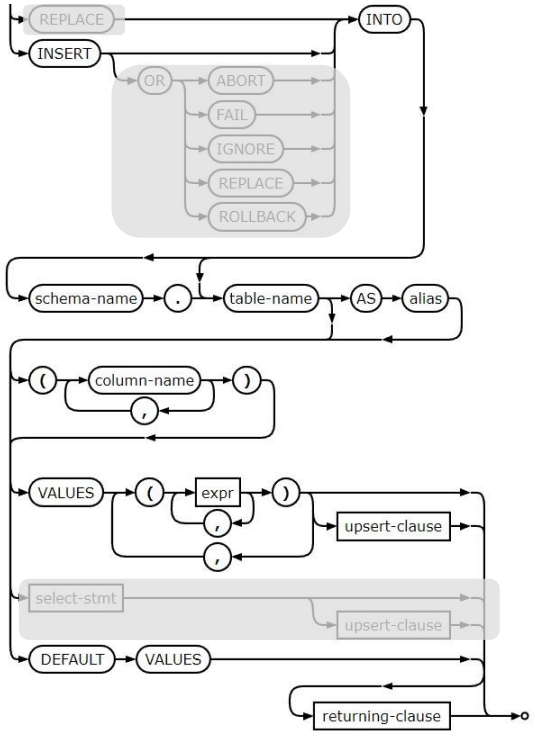
Les requêtes pour modifier les tables et leurs colonnes (avec SQLite)

Sur SQLite, la requête permet de nombreuses modifications qui changent les colonnes d'une table voire la table elle-même. Pour cela, on a d'abord | ALTER TABLE | qui permet de renommer la table avec | RENAME TO … |, de renommer une colonne | RENAME nom-colonne TO … |, d'ajouter des colonnes | ADD … | (en n'oubliant pas de spécifier son datatype après le nom de la colonne), et de supprimer des colonnes | DROP … |. Dans un second temps, on a aussi la requête | DROP TABLE … | qui permet de supprimer la table entièrement (⚠ cette requête est dangereuse et peut parfois nécessiter sur certaines gestions de bases de données d'une autorisation).



Les requêtes pour modifier / naviguer les valeurs de la table (avec SQLite)

En continuant, on va maintenant aborder les requêtes SQL qui permettre d'ajouter des informations et modifier les valeurs du tableau en commençant par | INSERT INTO | qui permet d'ajouter des valeurs dans le tableau selon les colonnes choisies. De même, je peux ajouter le nombre de ligne qu'on veut en une seule commande en continuant à ajouter des | VALUES |.




```sql
INSERT INTO recipies (
    title,
    slug,
    content,
    duration,
    online,
    create_at,
    email,
    author
    ) VALUES (
    'Sushi',
    'recette-sushi',
    'Etape 1 : prendre un contenant en y ajoutant
250g de farine, 3 cuillères à soupe
        d''huile d''olive, du sel et du poivre ;
Etape 2 : mélanger en y incorporant au fur à
mesure 50cl
        d''eau ; Etape 3 : attendre minimum 1h ;
Etape 4 : cuire la socca
        selon vos outils de cuissons (four à
pizza, four classique, poële, ...)',
    35,
    TRUE,
    1654723810,
    'mathuww@darnoguilem.com',
    'Mathuww'
);
```

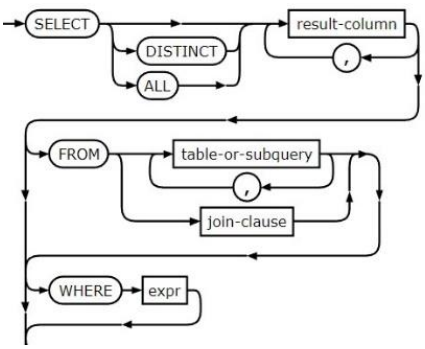
- *Le **TIMESTAMP** est un nombre qui permet de retranscrire la date entièrement selon l'année, le mois, le jour, l'heure, la minute et la seconde. Très important de même, le symbole « * » récupère toute les données possibles dans les conditions données.*

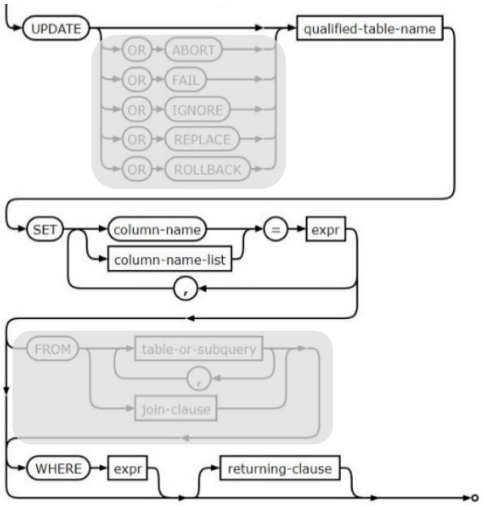
Mais à quoi ça sert d'ajouter des informations sans les afficher ? C'est dans ce contexte qu'on peut utiliser la requête SQL de sélection | SELECT |. C'est la requête la plus complète et importante de SQL, donc il faut bien se l'approprier, surtout sur les nombreuses conditions (en faisant attention à l'ordre des propriétés en ajoutant des parenthèses dans le cas où il y a plusieurs conditions) comme :

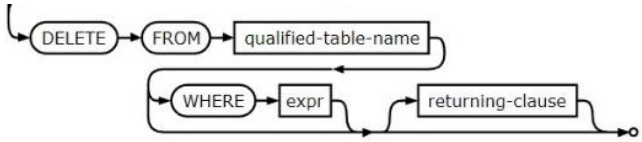
- Égalité ou une Vérité avec | `WHERE title = 'Sushi'` | ;
- Inégalité ou une Contrevérité avec | `WHERE duration != 45` | ;
- Supérieur ou Inférieur et/ou Égal avec | `WHERE duration <= 35` | ;
- Intervalle avec | `WHERE duration BETWEEN 30 AND 48` | ;
- Parmi les valeurs proposées avec | `WHERE slug IN ('recette-sushi', 'recette-socca')` | ;
- Valide sur une des deux conditions au minimum (« ou ») avec | `WHERE title IN ('Sushi', 'Socca') OR duration =35` | ;
- Valide sur ses deux conditions (« et ») avec | `WHERE slug IN ('recette-sushi', 'recette-socca') AND duration <=35` | ;
- NULL ou non NULL avec | `WHERE content IS NOT NULL` | ou | `WHERE content IS NULL` |;

```sql
--Exemple d'un 'SELECT'
SELECT title, content, duration, create_at, author /* Les valeurs correspondantes à ses colonnes seront
affichées. De même, on a la possibilité de les remplacer par * pour afficher toutes les colonnes */
FROM recipies --La table qu'on cherche
WHERE duration != 35 AND duration <= 40; --Les conditions
```

```sql
--Exemple de DELETE
(supprimer une valeur, une ligne)
DELETE FROM recipies WHERE title = 'Sushi';

--Exemple de UPDATE
(remplacer les valeur de colonne.s selon des conditions)
UPDATE recipies SET content = NULL WHERE title = 'Socca';
```







- *Il est important d'ajouter des condition dans | DELETE | et | UPDATE | car sinon toutes les valeurs/lignes se suppriment ou qu'ils se remplacent par la même valeur dans une même colonne, même si dans la plupart des cas, on vous avertit de la requête dangereuse.*

- *De même, la valeur NULL est une valeur spéciale qui permet de représenter l'absence de valeur pour une colonne donnée. Son comportement peut varier d'une base de données à l'autre.*

Cependant, on a un problème lorsque l'on crée des enregistrements dans notre table. Il est important de pouvoir les identifier de manière unique pour les conditions de récupération des modifications et des suppressions. En effet, si on utilise le champ titre, il peut être amené à changer dans le futur. Il nous faut donc une valeur (unique) qui sera invariante tout au long de la vie de notre enregistrement. Pour remplir ce besoin, on pourra se baser sur les clés primaires. Cette clé primaire une colonne du tableau qu'on ajoute à la création du tableau avec des contraintes spécifique. De même, les contraintes des tableaux permettent d'ajouter un supplément qui permet soit de contraindre, d'automatiser la colonne désignée ou encore d'avertir l'utilisateur. On peut les ajouter les uns à la suite des autres. Par exemple, on a la possibilité d'ajouter:

- | `PRIMARY KEY` | : la contrainte qui permet de signifier que les valeurs de cette colonne ont chacun un identifiant unique ;
- | `AUTOINCREMENT` | : la contrainte permet d'ajouter une valeur automatique qui suit une logique croissant des lignes précédentes (ligne 1 : 1, ligne 2 : 2, ligne 3 : 3, ligne 4 : 4, ...) ;
- | `NOT NULL` | : la contrainte permet de signifier qu'on ne peut pas donner une valeur « NULL » dans cette colonne ;
- | `UNIQUE` | : la contrainte permet de signifier qu'on ne peut pas donner une valeur existante précédemment dans cette colonne ;
- | `DEFAULT` | : la contrainte permet de signifier après cette balise, la valeur par défaut (par exemple, on aura cette valeur si on ne donne pas de valeur quand on crée/modifie une ligne).

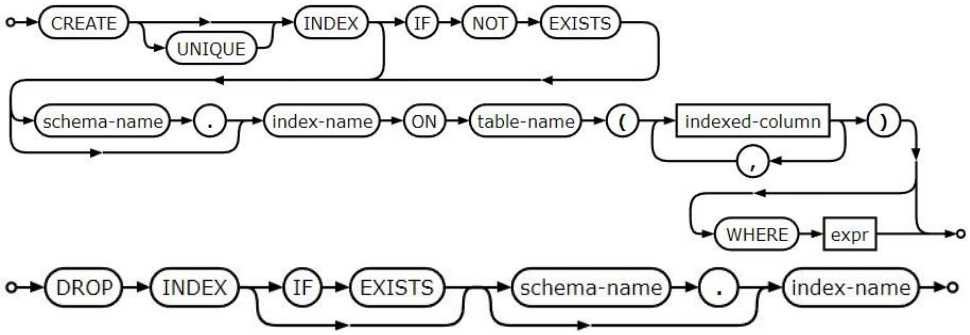
- *La contrainte | `AUTOINCREMENT` | ne prend pas en compte les lignes supprimées (ligne 1 : 1, ligne 2 : 2 => supprimer la ligne 2 égale à la valeur 2 => ajouter 2 autres lignes => ligne 1 : 1, ligne 2 : 3, ligne 3 : 4).*

```sql
-- Un exemple de CREATE TABLE avec une clé primaire
CREATE TABLE recipes (
    id INTENGER PRIMARY KEY AUTOINCREMENT,
    title VARCHAR(70),
    content TEXT,
    slug TEXT UNIQUE,
    duration SMALLINT,
    online BOOLEAN,
    create_at DATETIME,
    email TEXT,
    author VARCHAR(18)
);
```

Ces clés constituent un identifiant unique qui sera assigné à chaque ligne et qui permettra ensuite d'y faire référence plus simplement. Ces clés primaires offrent aussi l'avantage d'être indexée par le SGBD et permettent une récupération plus rapide lorsqu'elles sont utilisées dans les conditions. Il sera aussi possible de préciser que cette valeur s'incrémentera de manière automatique afin de ne pas avoir à préciser manuellement la clé lors de chaque insertion.

- *La contrainte | `EXPLAIN QUERY PLAN SELECT * FROM recipies WHERE id = 3` | permet de voir ce que la base donnée fait à chaque fois quand il récupère des valeurs. Avec cette requête, on peut savoir que la clé primaire et les index permettent de rechercher plus rapidement une valeur.*

En plus de la clé primaire, il sera possible d'indexer d'autres champs qui sont souvent utilisés comme condition de récupération. Par contre, les données indexées ont un coût plus élevé. On pourra aussi y adjoindre des contraintes d'unicité pour éviter les duplications.



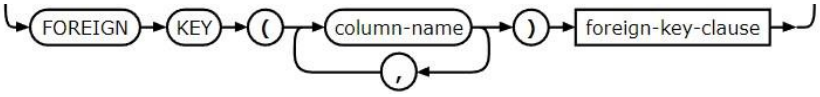

```sql
-- Des exemples de requêtes d'INDEX :
CREATE UNIQUE INDEX idx_slug ON
recipies (slug);
DROP INDEX idx_slug ;
```

Clés étrangères et jointures (avec SQLite)

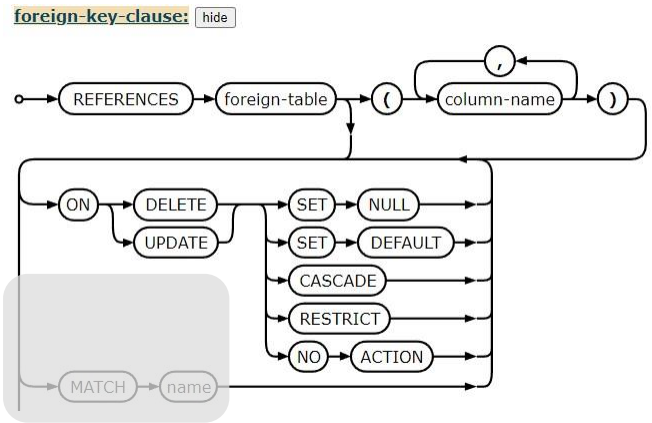
- *Pour intégrer l'option des jointures entre différents tableaux pour SQLite (exceptionnellement), il faut ajouter dès le début du code SQL, la requête* `PRAGMA foreign_keys = ON`.

Jusqu'à maintenant nous n'avons utilisé qu'une seule table pour expérimenter avec nos premières requêtes SQL. Dans la réalité on aura souvent besoin d'utiliser plusieurs tables pour représenter nos données que l'on pourra ensuite lier les unes aux autres grâce à des clés étrangères. C'est pour cela, on a besoin d'intégrer des clés étrangères dans le tableau où on veut intégrer le second tableau. Pour cela, j'ai fait un exemple de tableau de recettes s'appelant « recipies » et un tableau qui a informé si la recette est un dessert ou un plat s'appelant « categories ».

```
CREATE TABLE IF NOT EXISTS categories (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title VARCHAR(64) UNIQUE NOT NULL,
    description VARCHAR(255)
);

INSERT INTO categories (title)
VALUES
    ('Plat'),
    ('Dessert');

CREATE TABLE IF NOT EXISTS recipies (
    id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    title VARCHAR(64) UNIQUE NOT NULL,
    slug VARCHAR(50) UNIQUE NOT NULL,
    content TEXT,
    category_id INTEGER,
    FOREIGN KEY (category_id) REFERENCES categories (id) --Connexion de la clef étrangère de categories
);

INSERT INTO recipies (title, slug, category_id)
VALUES
    ('Gâteau au chocolat', 'gateau-chocolat', 2),
    ('Soupe', 'soupe', 1),
    ('Éclair au chocolat', 'eclair-chocolat', 2);
```



La nouvelle contrainte de la table

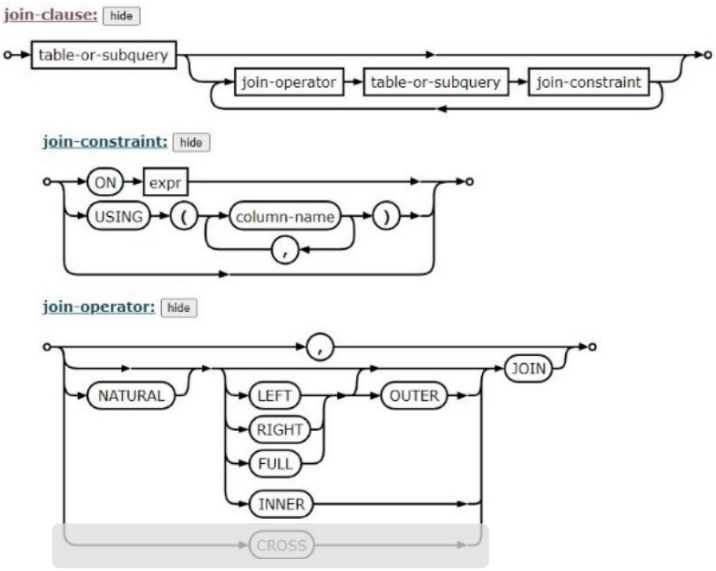


- ✓ **NO ACTION (valeur par défaut) :** lorsqu'une clé parent est modifiée ou supprimée de la base de données, aucune action particulière n'est entreprise.
- ✓ **RESTRICT :** empêcher la modification ou la suppression.
- ✓ **SET NULL :** remplacer la modification ou la suppression par la valeur « NULL ».
- ✓ **SET DEFAULT :** remplacer la modification ou la suppression par la valeur par défaut.
- ✓ **CASCADE :** lorsqu'une clé parent est modifiée ou supprimée de la base de données, chaque ligne de la table enfant qui était associée à la ligne parent supprimée/modifiée est également supprimée/modifiée.

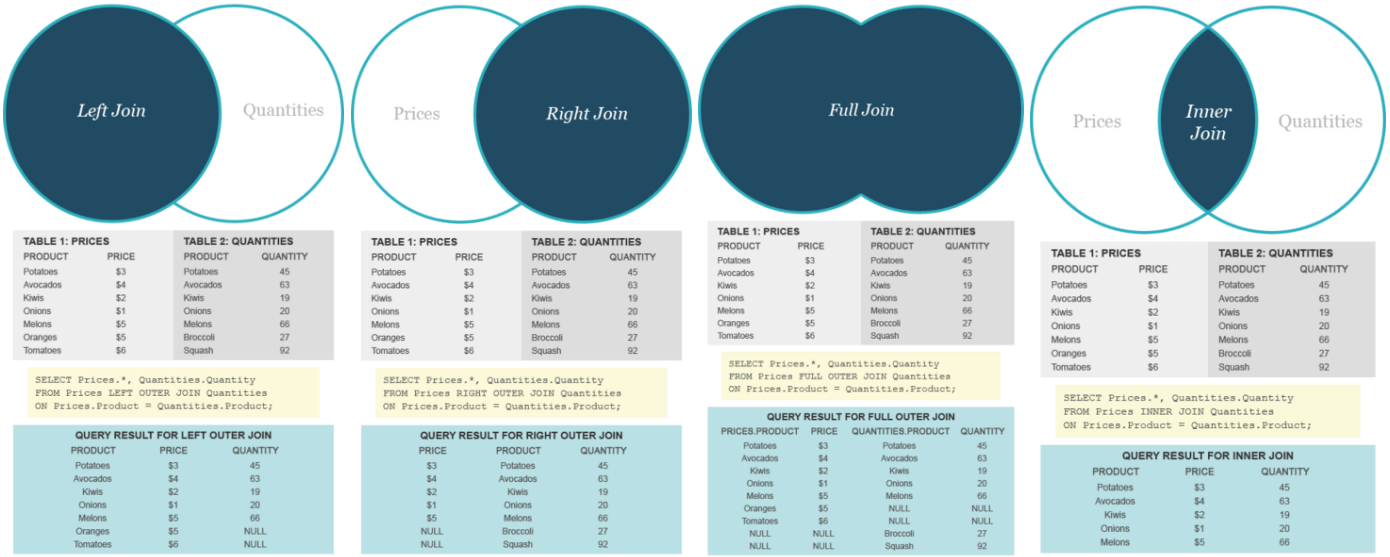
Rappelez-vous du schéma « SELECT », on avait l'option « join-clause ». Maintenant, on sait que cette option de sélection permet d'afficher la jointure des deux tableaux par clés étrangères. En SQL, une jointure est utilisée pour comparer et combiner - littéralement joindre - et renvoyer des lignes spécifiques de données à partir de deux tables ou plus dans une base de données. Une jointure interne trouve et renvoie les données correspondantes des tables, tandis qu'une jointure externe trouve et renvoie les données correspondantes et certaines données différentes des tables.

- *À partir du moment où tu utilises plusieurs tables, il faut utiliser la manière plus complexe d'écrire pour informer l'utilisation d'une colonne avec « nom de la table .(point) colonne ». On peut aussi faire un alias de la variable en le rajoutant juste après la variable. De même, on peut faire des alias pour la légende du tableau avec « AS ».*
- *Ne pas oublier que le « WHERE » s'utilise après le « JOIN ». Il faut se fier aux embranchements officiels.*

```
SELECT r.id, r.title, c.title AS category
FROM recipies r
INNER JOIN categories c ON r.category_id = c.id;
```

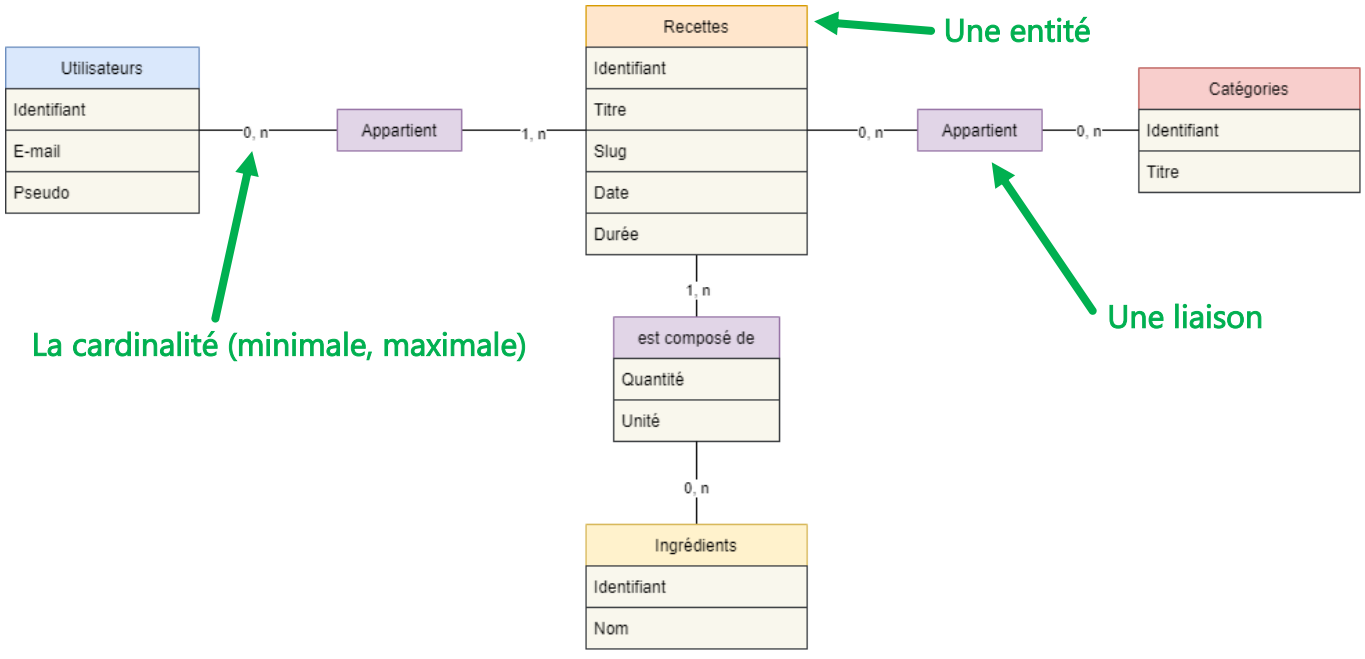


- ✓ **INNER (interne) :** une jointure interne recherche dans les tables des données correspondantes ou qui se chevauchent. Une fois trouvée, la jointure interne combine et renvoie les informations dans une nouvelle table.
- ✓ **LEFT OUTER (externe) :** une jointure externe gauche renverra toutes les données du tableau 1 et toutes les données partagées/correspondantes du tableau 2.
- ✓ **RIGHT OUTER (externe) :** une jointure externe gauche renverra toutes les données du tableau 2 et toutes les données partagées/correspondantes du tableau 1.
- ✓ **FULL OUTER (externe) :** une jointure externe complète, combine et renvoie toutes les données de deux tables ou plus, qu'il y ait ou non des informations partagées. Considérez une jointure complète comme une simple duplication de toutes les informations spécifiées, mais dans une table plutôt que dans plusieurs tables. Lorsque des données correspondantes sont manquantes, des valeurs nulles seront produites.



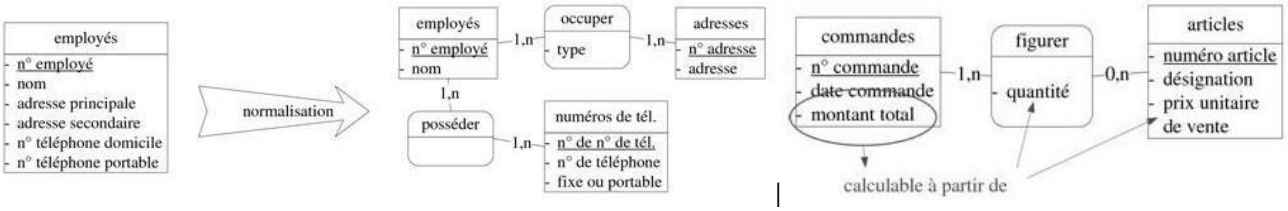
Le schéma MCD et MLD (avec draw.io)

Vu qu'on a vu la possibilité de lier les tables ensembles, il est temps de passer par la modélisation. Cette étape permet de réfléchir en amont à la structure de nos données et de concevoir plus facilement une base de données par la suite, car c'est important de réfléchir avant sur la structure de notre application. Cette schématisation suit des normes qui permettront une meilleure compréhension avec les autres développeurs sur le projet. La première a connaître est la norme MCD (dans notre langue, théorie) puis la norme MLD (représentative au code).

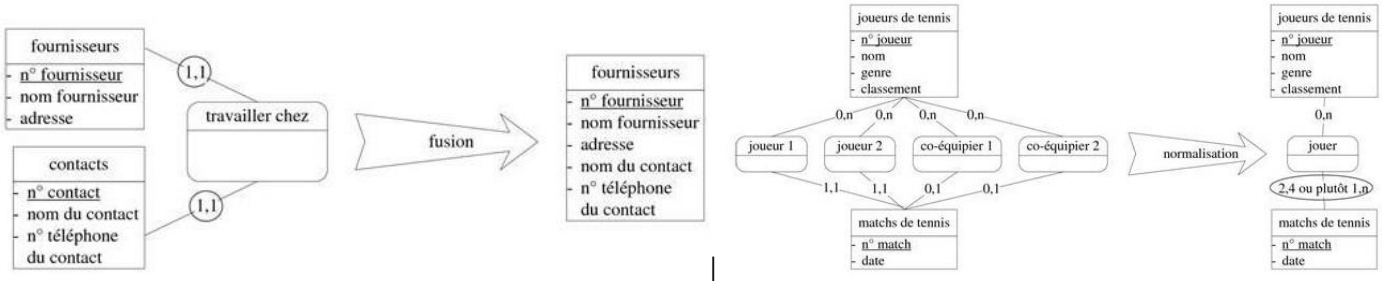


Les règles de normalisation MCD :

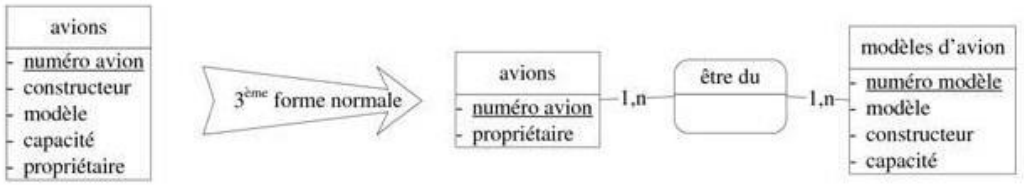
- ✓ Limiter/regrouper des entités similaires (*ex : une entité étudiants et une entité enseignants s'ils ont à peu près les mêmes colonnes => une entité personne ; ou encore limiter les colonnes identiques sur plusieurs entités comme l'adresse de livraison sur des entités clients et commandes*) donc deux entités homogènes peuvent être fusionnées ;
- ✓ Chaque entité doit posséder un identifiant (un entier au mieux) :
  - ➢ éviter les identifiants composés de plusieurs attributs (comme un identifiant formé par les attributs nom et prénom), car d'une part c'est mauvais pour les performances et d'autre part, l'unicité supposée par une telle démarche finit tôt ou tard par être démentie/contestée ;
  - ➢ préférer un identifiant court pour rendre la recherche la plus rapide possible (éviter notamment les chaînes de caractères comme un numéro de plaque d'immatriculation, un numéro de sécurité sociale ou un code postal) ;
  - ➢ préférer un identifiant court pour rendre la recherche la plus rapide possible (éviter notamment les chaînes de caractères comme un numéro de plaque d'immatriculation, un numéro de sécurité sociale ou un code postal) ;
- ✓ Phase de normalisation : remplacer les attributs en plusieurs exemplaires en une association supplémentaire de cardinalités maximales n et ne pas ajouter d'attribut calculable à partir d'autres attributs (qui faut retirer du schéma) :



- ✓ Normalisation des attributs : les attributs d'une association doivent dépendre directement des identifiants de toutes les entités en association (*ex d'après le premier schéma MCD : la quantité dépend de l'ingrédient et de la recette*) ;
- ✓ Normalisation des associations : éliminer les associations fantômes (schéma 1 : les cardinalités sont toutes 1, 1 ; ou schéma 2 : une association suffit pour remplacer les 4 associations pour participer) :



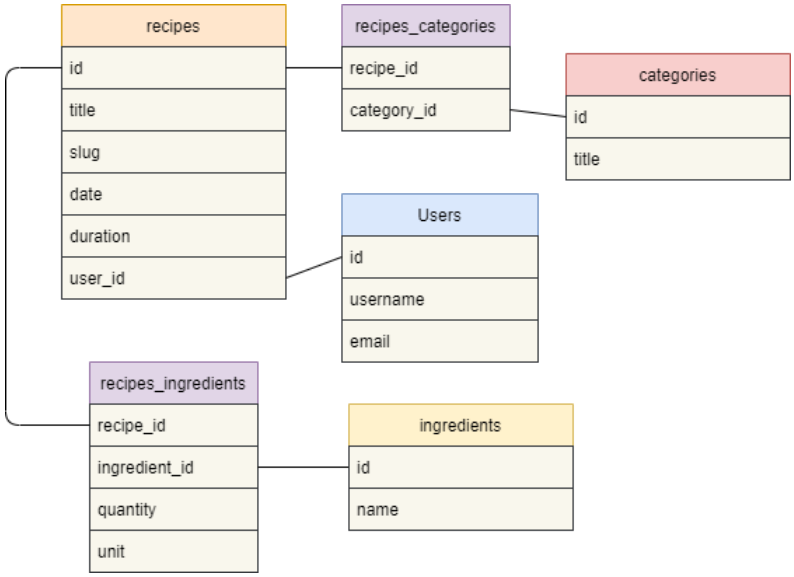
- ✓ Si jamais, des attributs d'une entité ne dépends pas directement de son identifiant, il faut placer d'autre attribut ressemblant dans une autre entité séparée, mais en association avec la première :



- ✓ Dans une entité, pour un individu, un attribut ne peut prendre qu'une valeur et non pas, un ensemble ou une liste de valeurs.

Les règles de normalisation MLD (s'inspirant au schéma MCD) :

- ✓ *L'écriture et les appellations se fait en général en anglais ;*
- ✓ Tous les entités du schéma MCD deviennent des tables ;
- ✓ Les relations de type 0, 1 devient une clé étrangère à la table principale ;
- ✓ Les relations de type 0, n ou 1, n ont une table de liaison qui contient aux moins une clé étrangère de chaque table reliée (qui s'écrit dans l'ordre alphabétique « nom de la première table (tiret du bas) nom de la seconde table ») ;
- ✓ Les relations de type 0, 1 devient une clé étrangère à la table principale avec les contraintes « UNIQUE » et « NOT NULL ».

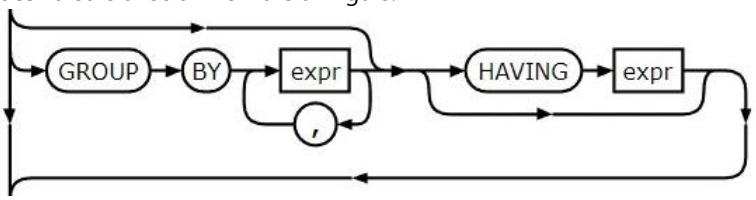


Agréger les données (avec SQLite)

L'agrégation va permettre de fusionner des lignes ensemble afin de récupérer des informations intéressantes (moyenne, total, somme, ...). Pour cela, il existe de multiples fonctions utilisées dans la requête SELECT comme :

- `AVG(X)` : système pour effectuer une moyenne (utile pour un système de notation) ;
- `COUNT(X)` ou `COUNT(*)` : système pour compter le nombre d'éléments de la recherche ;
- `GROUP_CONCAT(X, ', ')` : système pour mettre bout à bout toutes les valeurs par le symbole souhaité ;
- `MIN(X)` ou `MAX(X)` : système pour chercher la valeur minimale/maximale ;
- `SUM(X)` : système pour chercher le total (l'addition) des valeurs ;
- `TOTAL(X)` : système pour chercher le total (l'addition) des valeurs avec un nombre à virgule.

```
SELECT SUM(duration) FROM recipes;
SELECT GROUP_CONCAT(title, ' ; ') FROM recipes;
```



Ces fonctions (dans « SELECT », le schéma ci-dessus représente la suite de cette requête SQL) peuvent être utilisé avec la commande « GROUP BY » qui permet de grouper les éléments en fonction d'une ou plusieurs colonnes. On ne peut l'utiliser qu'à partir de la première agrégation concernée en allant vers la droite (jamais à gauche). Lors d'une agrégation, on ne pourra sélectionner que les colonnes qui sont dans le « GROUP BY » et celles qui utilisent les fonctions d'agrégation. Cependant, avec « GROUP BY » et donc l'agrégation, on n'utilise pas « WHERE » mais « HAVING » (la même propriété et la façon d'écrire).

```
SELECT COUNT(ir.recipe_id) as counter, i.name, r.duration
FROM ingredients i
LEFT JOIN ingredients_recipes ir ON ir.ingredient_id = i.id
LEFT JOIN recipes r ON r.id = ir.recipe_id
GROUP BY i.name, r.duration
HAVING counter >= 1;
```

En plus de la fonction d'agrégation, il est aussi possible d'éviter les doublons à l'aide de l'instruction « DISTINCT » dans les requêtes « SELECT ». Cela pourra s'avérer utile lorsque l'on utilisera les jointures qui ont tendance à générer beaucoup de lignes en double.

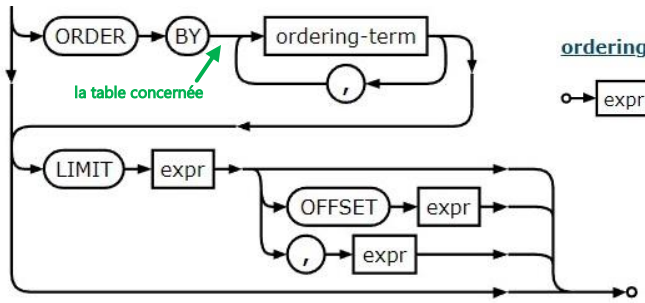
```
SELECT DISTINCT i.name
FROM ingredients i
LEFT JOIN ingredients_recipes ir ON ir.ingredient_id = i.id
LEFT JOIN recipes r ON ir.recipe_id= r.id
WHERE ir.recipe_id IS NOT NULL;
```

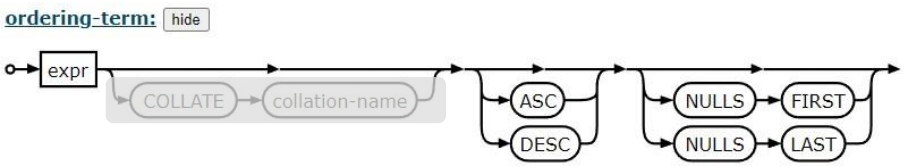
Organiser les données (ORDER et LIMIT avec SQLite)

Ce serait bien maintenant (après d'avoir vu comment conter les valeurs), de classer les valeurs, d'organiser les valeurs. C'est dans ce but que nous allons voir comment organiser et limiter le nombre de résultats que l'on obtient. Pour ce mettre au clair, c'est que des nouvelles options à ajouter de la requête SELECT (où on voit la suite dans le premier schéma ci-dessous). D'abord, pour organiser les données, on pourra se reposer sur la commande « ORDER BY ». Sinon, on pourra aussi limiter le nombre d'enregistrement que l'on récupère à l'aide de la commande « LIMIT » (qui ont deux possibilités de notation mais la plus évidente est

`LIMIT exp` ou `LIMIT exp OFFSET exp`.

- *Ne pas confondre des nombres en entier (qui s'organise comme en math) et des nombres en chaînes de caractères (qui s'organise différemment/alphabétiquement ; par exemple avec 10, 20, 112, la liste s'organise en '10', '112', '20').*






```
SELECT COUNT(ir.recipe_id) as counter, i.name
FROM ingredients i
LEFT JOIN ingredients_recipes ir ON ir.ingredient_id = i.id
GROUP BY i.name
ORDER BY counter DESC NULLS FIRST, i.name ASC
LIMIT 3 OFFSET 1;
```

Requêtes imbriquées (avec SQLite)

Ba, maintenant, on est arrivé au stade où il faut des requêtes dans des requêtes avoir des bon résultats. En effet, quand on a beaucoup de table éloigné, c'est conseillé de savoir comment utiliser les requêtes imbriquées. Cela permet d'utiliser le résultat d'une requête à différents niveaux. Cependant, cette méthode prend/pèse beaucoup de ressource à la recherche (impact assez important sur les performances). Il ne faut pas le mettre partout, surtout quand la requête imbriquée est corrélée (en bref, c'est-à-dire qu'elle utilise des tables, des valeurs de la requête principale). De même, il ne faut pas hésiter d'utiliser « Explain Query Plan » devant la requête. On peut le faire sous trois façon (toujours entourées de parenthèses).

✓ Dès le début après le SELECT (exemple : récupérer les recettes avec le nombre d'ingrédients attribuées)

```
SELECT *, (
    SELECT COUNT(*) FROM ingredients_recipes
    WHERE recipe_id = r.id
) AS number_ingredients
FROM recipes r;
```

✓ Après le FROM (exemple : récupérer le nombre de recettes) => méthode utilisée rarement

```
SELECT r.count
FROM (
    SELECT COUNT(id) as count FROM recipes
) AS r;
```

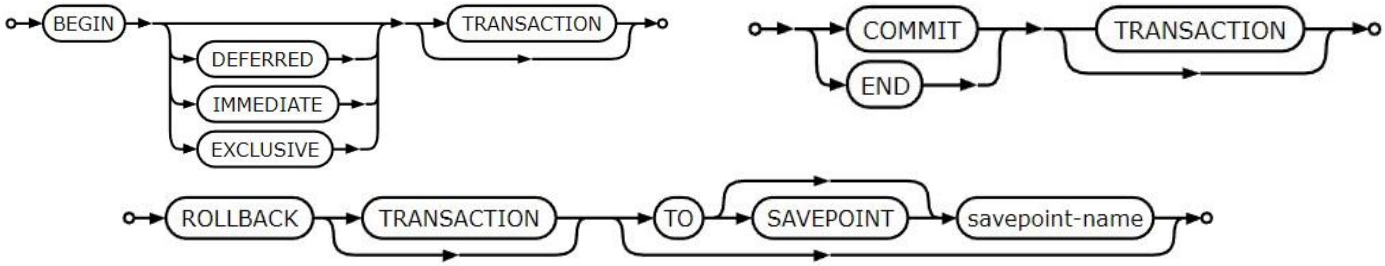
✓ Dans les conditions, après WHERE en n'oubliant pas « IN » (exemple : récupérer les ingrédients qui ont la catégories 'Dessert')

```
SELECT i.*
FROM ingredients_recipes ir
LEFT JOIN ingredients i ON i.id = ir.ingredient_id
WHERE ir.recipe_id IN (
    SELECT cr.recipe_id
    FROM categories c
    LEFT JOIN categories_recipes cr ON c.id = cr.category_id
    WHERE c.title = 'Dessert'
);
```

- *Une requête imbriquée n'est pas corrélée (en lien avec la requête principale) par défaut.*

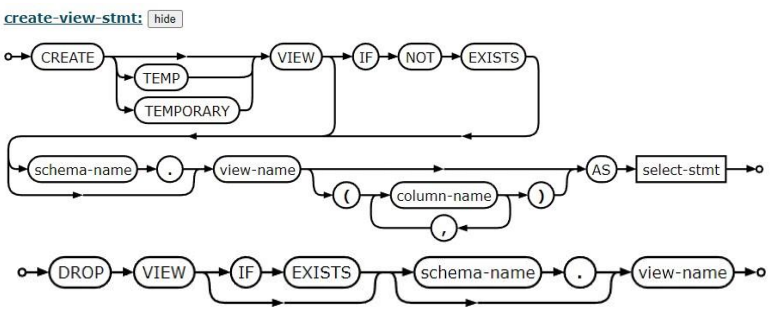
Autres fonctionnalités avec SQLite pour sécuriser, optimiser et préciser son code (Transactions, Vues et Triggers)

Tout d'abord, dans un code SQL, nous avons tous peur de perdre de nombreuses données quand on se trompe sur certaines requêtes. Pour cela, SQL possède les transactions qui sont une mécanique qui permet de grouper l'exécution de plusieurs requêtes afin de pouvoir revenir en arrière en cas de problèmes. On commencera par activer la transaction avec `BEGIN TRANSACTION;`. Puis on exécutera ensuite une liste de requêtes SQL qui n'affecteront pas directement la base de données. On devra terminer la transaction à l'aide d'un `COMMIT TRANSACTION;`. Sauf que tant qu'on ne fait pas un « COMMIT », on a toujours la possibilité aussi d'annuler une transaction en revenant en arrière à l'aide d'un `ROLLBACK TRANSACTION;`. C'est une méthode utile pour tester des tables et surtout des clés étrangères. De même, quand il y a un conflit, sur SQLite, le ROLLBACK s'active automatiquement.



De surcroît, nous avons un outil que je trouve trop cool pour ce langage. Ce sont les vues qui permettent de créer une table virtuelles à partir du résultat d'une requête SQL. Les vues seront nommées ce qui permettra d'y faire référence plus facilement. C'est une sorte de SELECT attribué à une table. Par contre, les merveilles s'accompagnent toujours d'un coût important sur les performances, surtout quand la requête a des jointure, des requêtes imbriquées et etc. Par exemple,

```
CREATE VIEW recipes_with_ingredients
AS
 SELECT r.title, GROUP_CONCAT(i.name, ', ') AS
ingredients
 FROM recipes r
 LEFT JOIN ingredients_recipes ir ON ir.recipe_id =
r.id
 LEFT JOIN ingredients i ON i.id = ir.ingredient_id
 GROUP BY r.title;
 SELECT *
 FROM recipes_with_ingredients
 WHERE ingredients LIKE '%Farine%';
```

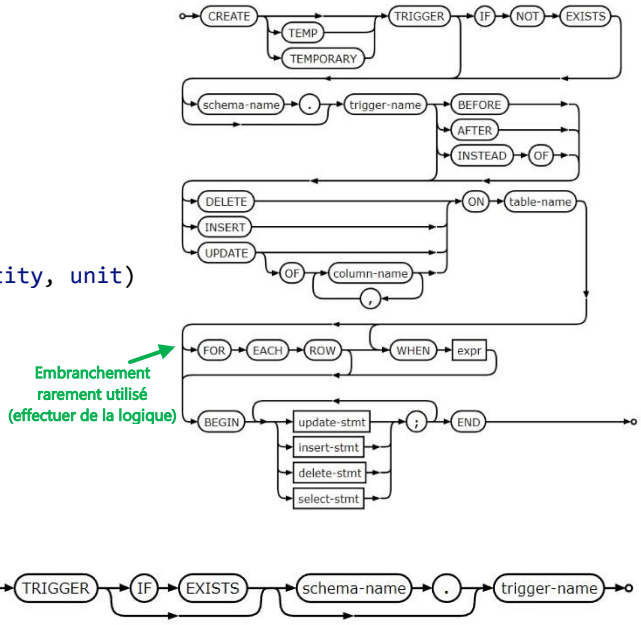


Pour finir en beauté avec SQLite, on va aborder les « **TRIGGERS** » qui permettent de rajouter de la logique lorsque certaines opérations sont effectuées sur la base de données. De même, avant de montrer un exemple, il faut faire la différence des notions « **NEW** » et « **OLD** » s'écrivant « NEW.column-name » ou « OLD.column-name » :

- ✓ « INSERT » ➔ « NEW » fait référence à la valeur qui s'est ajoutée,
- ✓ « UPDATE » ➔ « NEW » fait référence à la nouvelle valeur, et « OLD » fait référence aux données avant la mise à jour,
- ✓ « DELETE » ➔ « OLD » fait référence à la ligne supprimée.

Cela reste un outils et une option puissant(e) mais qui possède aussi un coût énorme sur les performances. Revenons dans nos données de recettes de cuisine, on peut utiliser les « TRIGGER » pour sauvegarder le nombre de fois qu'un ingrédient est utilisé.

```
CREATE TRIGGER increment_usage_count_on_ingredients_linked
AFTER INSERT ON ingredients_recipes
BEGIN
    UPDATE ingredients
    SET usage_count = usage_count + 1
    WHERE id = NEW.ingredient_id;
END;

SELECT * FROM sqlite_master;

INSERT INTO ingredients_recipes (recipe_id, ingredient_id, quantity, unit)
VALUES
    (1, 7, 10, 'g');

CREATE TRIGGER decrement_usage_count_on_ingredients_unlinked
AFTER DELETE ON ingredients_recipes
BEGIN
    UPDATE ingredients
    SET usage_count = usage_count - 1
    WHERE id = OLD.ingredient_id;
END;

SELECT * FROM sqlite_master;

DELETE FROM ingredients_recipes WHERE recipe_id = 1 AND ingredient_id = 7;
```



Embranchement rarement utilisé (effectuer de la logique)

- *La requête* `SELECT * FROM sqlite_master;` *permet d'afficher la table cachée de sqlite qui permet d'afficher/vérifier tous les grands ajouts d'une base de données sqlite (le nombre de table, d'index, de triggers, …).*
- *Si un trigger ne s'exécute pas sur VS code, il faut utiliser le terminal en commençant par « ./sqlite » + TAB + « nom de la base de donnée ». De même pour quitter, on utilise « .quit » ou encore « .help » pour de l'aide. Par contre, il faut faire attention à ne pas commettre des erreurs car le terminal ne les avertie pas.*

Un autre système de bases de donnée : MySQL

Depuis le début de cette formation, nous avons utilisé SQLite. Même si c'est une bonne base de données pour commencer, ce n'est pas forcément la base de données que vous allez utiliser pour vos premiers projets professionnels. Je vous propose aujourd'hui de découvrir rapidement les particularités de **MySQL**, un système de gestion de bases de données que vous allez très souvent retrouver, surtout si vous travaillez avec des hébergements mutualisés dont la facilité à l'administrer ou si vous voulez plus de fonctionnalités et plus de cadres comme les types strictes. De plus, on a la possibilité d'avoir une architecture plus adaptée pour le client/serveur (comme avec PHP). L'avantage de cela d'être un célèbre langage est qu'il y a beaucoup plus de possibilité, plus de communautés pour les problèmes à résoudre, etc.

Après l'installation du serveur de la base de donnée et du shell MySQL (en vérifiant qu'ils sont dans les variables d'environnement du système Windows), on va juste activer l'extension « MySQL » de WeiJan Chen. Puis on ouvre l'onglet Database et on ajouta une connexion avec le nom du serveur et de notre mot de passe MySQL (en n'oubliant pas de décocher « Hide System Shema »).

- *Ce qui est utile est que MySQL est très célèbre. Il permet d'avoir de l'aide plus facilement surtout dans les forums. Par contre, il faut faire attention de ne pas se tromper sur des informations périmées.*
- *Le « AUTOINCREMENT » devient « AUTO_INCREMENT » sous MySQL.*
- *Une nouvelle syntaxe interactive sous MySQL sur le « INSERT INTO » et « UPDATE» qui est légèrement plus lisible quand il y a beaucoup de valeurs :*

```
INSERT INTO [table] SET column = value, column2 = value WHERE [condition];
UPDATE [table] SET column = value, column2 = value WHERE [condition];
```

- *Information utile : Maria DB est un variant de MySQL qui s'est créé pour d'ordre politique par son créateur. Il a juste aujourd'hui quelques informations précises sur les stockages en plus.*

La première différence de MySQL : les différents DATATYPES

En effet, MySQL a plus de datatypes spécifiques mieux adaptés aux valeurs avec des fonctions plus spécifiques. Par contre, ce n'est pas qu'un avantage, car MySQL plus strictes sur le choix de ses datatypes (on ne peut pas mettre d'autres valeurs que le datatype propose). Les nouveautés sont… (pour rappel, tous les autres datatypes vu sur SQLite, sont intégrés dans MySQL)

Exemple de DATATYPES de MySQL	*Le type de résultat des valeurs*
1. **DECIMAL(P, D)** ou **DEC(P, D)** ou **FIXED(P, D)** ou **NUMERIC(P, D)** → = valeur exacte → P est la précision qui représente le nombre total de chiffres significatifs (de 1 à 65) → D est l'échelle qui représente le nombre de chiffres après la virgule (de 0 à 30)	**NUMERIC** (*conversion en REEL ou INTENGER*)
2. **BOOLEAN** → **true ou false** (1 ou 0) est remplacé automatiquement par un **TYNYNT(1)**	
3. **VARBINARY (N)** → similaire au type VARCHAR, mais stocke des chaînes binaires jusqu'à 255 octets plutôt que des chaînes de caractères non binaires	**BINARY**
4. **TINYBLOB** → une colonne BLOB d'une longueur maximale de 255 octets	
5. **MEDIUMBLOB** → une colonne BLOB d'une longueur maximale de 16 777 215 octets	
6. **LONGBLOB** → une colonne BLOB d'une longueur maximale de 4 294 967 295 octets	

- *Plus d'information sur les datatypes MySQL :* https://dev.mysql.com/doc/refman/8.0/en/data-types.html.

De même, on peut ajouter des contraintes sur les DATATYPES de type INTENGER qui permet de choisir si on veut des nombres négatifs et positifs avec **SIGNED** (contrainte par défaut) ou que des nombres positifs avec **UNSIGNED**. L'avantage de ce procédé quand on choisit Un est d'avoir plus de nombres disponible (au lieu d'un octet moitié positif et moitié négatif, on aura un octet entier positif). Par exemple, la possibilité des valeurs de datatype *TINYINT* passe de [-127 – 128] à 350.

7. **DATE** → sauvegarder un jour au format "YYYY-MM-DD"	**TIME**
8. **TIME** → sauvegarder un temps "hh:mm:ss"	
9. **DATETIME** → sauvegarder votre jour et votre temps "YYYY-MM-DD hh:mm:ss"	
10. **TIMESTAMP** → la même chose qu'un « DateTime » mais utilise un timestamp en interne (il fonctionne selon le fuseau horaire donné donc par un traitement). → La valeur zéro/minimal sera le 1er janvier 1970, → La valeur maximal sera la dernière seconde du dernier jour de 2038 (32 octets).	
11. **YEAR** → permet de sauvegarder une année	

Alors les **DATATYPES de temps**, ils ont l'avantage d'être plus utiles sur MySQL par ses multitudes de fonctions facile à utiliser (https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html) comme | YEAR (…) |, ou | CONVERT_TZ (datetime, le fuseau d'horaire d'origine, le nouveau fuseau d'horaire) | qui permet de convertir une date (sous une timezone particulière), ou |TIMEDIFF(temps_n°1, temps_n°2)| ou encore | NOW |. On a aussi la possibilité de changer le fuseau horaire de la base de données avec | SET time_zone = '+00:00' ; |. On peut de même récupérer le fuseau horaire de la base de donnée avec | SELECT @@global.time_zone, @@session.time_zone ; |. On peut de même dans la création d'une colonne de DATATYPES de type « TIME », rajouter | … NOT NULL DEFAULT CURRENT_TIMESTAMP [ON UPDATE CURRENT_TIMESTAMP] ; |.

- ***UTC time*** *signifie du temps qui n'est pas traité par un quelconque langage, programme, base de données car il a déjà été traité par le programmeur.*
- *On peut savoir le fuseau horaire de la base de données détaillé avec | SELECT TIMEDIFF(NOW(), CONVERTZ_TZ(NOW(), @@session.time_zone, '+00:00')) ; |.*
- *On a la possibilité d'ajouter le « TIMESTAMP » de la base de donné par défaut dans une table avec | … NOT NULL DEFAULT CURRENT_TIMESTAMP ; |. Puis en finissant par | … [ON UPDATE CURRENT_TIMESTAMP] |, à chaque mise à jour, le « TIMESTAMP » de la base de données se modifiera. Par contre, il faut faire attention aux fuseaux horaires de la base de données.*

12. **GEOMETRY** → le parent de tous les types spatiaux qui sont tous cryptés en binaires, il a de nombreux paramètres mais il est trop compliqué	**SPATIAL**
13. **POINT** → une géométrie à zéro dimension qui représente un emplacement unique dans l'espace de coordonnées	
14. **POLYGON** → une surface plane représentant une géométrie à plusieurs côtés (cas particulier comme redéfinir des régions ou encore les départements français)	
15. **MULTIPOINT** → une collection de géométrie composée d'éléments « Point » qui ne sont pas connectés ou ordonnés de quelque manière que ce soit (par exemple dans le monde, elle peut représenter une archipel de petites îles)	

Alors les **DATATYPES spatial** (pour une utilisation simple et non pro), pareil que celle du temps, ils sont utiles principalement pour récupérer des géolocalisations par leurs fonctions faciles à utiliser (https://dev.mysql.com/doc/refman/8.0/en/spatial-function-reference.html) comme | ST_GeomFromText ('[DATATYPES] (altitude longitude) ') |, ou encore | ST_Distance_Sphere() |.

- On ne peut pas récupérer manuellement les données spatiales vu qu'elles sont binaires.

```
INSERT INTO information (title, location) VALUES
('Poitiers', ST_GeomFromText('POINT(0.340196 46.580260)')),
('Montpellier', ST_GeomFromText('POINT(3.876734 43.611242)'));

SELECT CONCAT(ROUND(
    ST_Distance_Sphere(
        (SELECT location FROM information WHERE title = "Nice"),
        (SELECT location FROM information WHERE title = "Toulouse")
    ) /1000), ' km') AS distance_Nice_Toulouse;

SELECT title FROM information WHERE ST_Distance_Sphere(
    (SELECT location FROM information WHERE title = "Montpellier"),
    location) > 300000;
```

16. **JSON** → stocke les documents JSON dans un format interne qui permet un accès rapide en lecture aux éléments du document. Le format binaire JSON est structuré de manière à permettre au serveur de rechercher des valeurs dans le document JSON directement par clé ou index de tableau, ce qui est très rapide. → Le format binaire JSON est structuré de manière à permettre au serveur de rechercher des valeurs dans le document JSON directement par clé ou index de tableau, ce qui est très rapide. → Le stockage d'un document JSON est approximativement le même que le stockage de données LONGTEXT.	**JSON**

Alors le **DATATYPES JSON**, avant d'en parler, le format JSON est un **langage simple où la machine et les humains se repèrent facilement dans un texte** (un objet qui commence par des accolades ou un tableau qui commence par des crochets). C'est un des seul fonctionnalité MySQL qui est totalement différent de MariaDB par ses fonctions simple et utile comme | JSON_SET(colonne, chemin vers la propriété, la nouvelle valeur) |.

- *Quand on met une valeur indexée, ce DATATYPES JSON prend beaucoup de ressources (en binaires).*
- *Pour chercher une valeur/chemin sur un fichier JSON sous MySQL, il faut commencer par | colonne->"$.propriété" | et continuer par des points et des propriétés pour aller en profondeur dans le format JSON. De même, avec une double flèches | colonne->>"$.propriété" | permettent d'enlever les caractères qui évitent les confusions, comme le double slash « // » qui permet d'ajouter des guillemets (le contraire de la simple flèche qui affiche la valeur brute).*

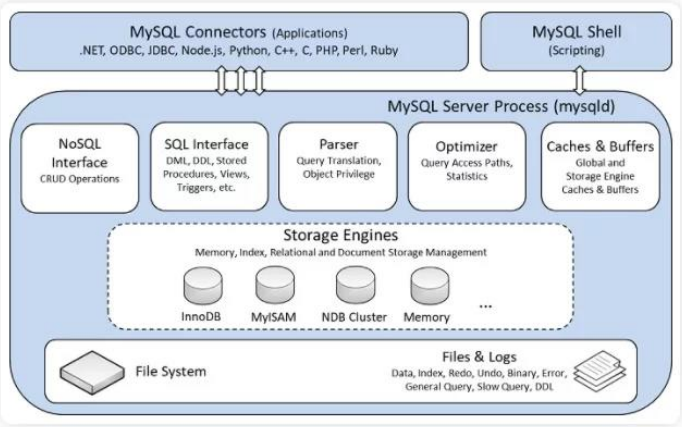
```
INSERT INTO information SET author = '{"name": "John\\"Doe", "age": 20}';
SELECT author->>"$.name" FROM information;
UPDATE information SET author= JSON_SET(author, '$.age', 22) WHERE id=1;
```

Les différentes contraintes propre à MySQL

Maintenant, qu'on a bien vu le langage SQL avec SQLite, on peut rentrer en profondeur sur la structure de MySQL. Une base de donnée a plusieurs outils dont MySQL Connectors (ses applications), plusieurs interfaces selon nos besoins et son Shell. Et surtout, elle a différents moteurs (storage engine) qui permettent de faire des opérations différentes avec sa base de donnée. Doit-on choisir un moteur plus rapide ou doit-on choisir un moteur plus libre avec plus de fonctionnalités. Pour cela, je vais comparer les deux plus moteurs connus : **InnoDB** et **MyISAM**.

- *De même, pour savoir quelle moteur notre base de donnée utilise, il faut utiliser la requête | SHOW ENGINES; | et on peut changer le moteur d'une table avec | ALTER TABLE `table` ENGINE=MYISAM; |.*

InnoDB	MyISAM
*un moteur assez performant et plus lent faisant partie de la famille des moteurs **transactionnels** (Il s'assure que les relations entre les données de plusieurs tables sont cohérentes et que si l'on modifie certaines données, que ces changements soient répercutés aux tables liées)*	*un moteur **non transactionnel** assez rapide en écriture et très rapide en lecture (ceci vient en grande partie du fait qu'il ne gère pas les relations ni les transactions et évite donc des contrôles gourmands en ressources mais perd en sûreté ce qu'il gagne en vitesse)*
est la meilleure option lorsque vous utilisez une base de données plus grande car elle prend en charge les transactions et le volume	*convient aux petits projets*
*MyISAM et InnoDB ont sensiblement la **même performance en lecture**, par contre **en écriture, InnoDB est plus lent que MyISAM**. De plus, les données stockées avec **InnoDB occupent plus de place sur le disque que MyISAM**.*	
Une fois qu'une table est supprimée, elle ne peut pas être rétablie	*Une fois qu'une table est supprimée, elle peut être rétablie*
le champ AUTO_INCREMENT fait partie de l'index	
*prend en charge les contraintes d'intégrité référentielle **FOREIGN-KEY, gestion des relations, supporte les transactions***	*ne prend pas en charge les contraintes d'intégrité référentielle **FOREIGN-KEY**, **pas de gestion de relation**, **ne supporte pas les transactions***
le verrouillage au niveau des lignes est possible et il y'a un verrouillage au niveau de la table	*il n'y a aucune possibilité de verrouillage au niveau des lignes, d'intégrité relationnelle il y'a un verrouillage au niveau de la table*
*Depuis MySQL 5.6.4, **les deux moteurs supporte l'index FULLTEXT***	
***l'insertion et la mise à jour sont beaucoup plus rapides** que MyISAM*	***l'insertion et la mise à jour sont beaucoup plus lentes** que InnoDB*



Autres moteurs de la base de données MySQL :

Memory : stocke les données non pas dans des fichiers sur le disque dur du serveur MySQL, mais dans la mémoire RAM. Cela permet un gain très important de performance, mais en contrepartie, il n'y a pas de persistance des données (quand on arrête le serveur MySQL, on perd toutes les données contenues dans les tables MEMORY). Ce moteur peut être utilisé pour stocker les utilisateurs connectés à un site internet par exemple (ces informations sont par nature éphémères, donc on n'a pas besoin de persistance).

Blackhole : est comme son nom l'indique un trou noir. C'est une sorte de puits sans fond, rien de tout ce que vous écrirez dans une table blackhole ne sera stocké. À première vue cela n'a aucun intérêt, mais on peut s'en servir pour faire des tests (benchmarks…).

CSV : enregistre les données dans des fichiers CSV (fichier texte avec chaque colonne séparée par un point-virgule (;), lisible dans un tableur comme Excel.

Archive : est spécialement conçu pour enregistrer des données qui ne changent pas, et pour optimiser le stockage en compressant les données de la table. On ne peut faire que des INSERT et des SELECT sur une table archive, pas d'UPDATE. Ce moteur est en général utilisé pour stocker des logs (journaux).

Merge (*alias MRG_MyISAM*) : est très proche de MyISAM, sauf qu'il permet de stocker une table dans plusieurs fichiers différents. Cela permet d'améliorer les performances sur de très grosses tables, et de compresser les données anciennes (sur lesquels il n'y aura plus d'écriture) avec myisampack.

Donc, vous vous demandez mais c'est quoi le lien avec les contraintes. Ba, en fait, on peut en plus de mettre à jour un moteur sur une table ; on peut aussi créer une table de toute pièce avec le moteur voulu avec la contrainte | ENGINE= ; |.

L'autre contrainte permettant de varier les paramètres fondamentaux est le « **Character Set** ». Il a pour objectif de changer l'encodage de la base de donnée, d'une table. Par exemple, on peut avoir l'encodage latin, chinois ou encore des encodages uniformes comme utf8. On peut connaître l'encodage de votre serveur avec | SHOW VARIABLES LIKE 'character_set_server'; | et celle de la base de données | SELECT default_character_set_name FROM information_schema.SCHEMATA WHERE schema_name = "tuto"; |. Par contre, c'est conseillé de laisser l'encodage par défaut de la base de données ou celle du serveur (si c'est le seul encodage). On a principalement 4 « Character Set » à savoir :

- « *ascii* » → basic 7-bit codes,
- « *latin1* » → ascii, avec les caractères dont l'Europe de l'Ouest a le plus besoin,
- « *utf8* » → le 1er, le 2ème et le 3ème octet de l'utf8 (incluant les émojis et les caractères chinois),
- « *utf8mb4* » → l'ensemble complet des caractères UTF8, couvrant toutes les langues actuelles.

Et ce qui va avec les « Character Set » est la manière de classer, de les sélectionner les caractères, les strings. On appelle cela le classement ou encore le « **Collation** ». Cela permet de savoir dans les requêtes de recherche si a=a ou/et a=A. Il y a de nombreux paramètres selon les « Character Set » selon si c'est sensible à la casse ou insensible à la casse (si on peut traiter quelques caractère en minuscule pour avoir de meilleur recherche). L'un se finit par « **_cs** » et l'autre par « **_ci** ». Par contre, on a la possibilité d'utiliser la requête | SELECT BINARY ... ; | qui permet de comparer les valeurs sur leurs nombres octets au lieu de leur similitude, très pratique pour les mots de passe (par exemple, avec un « SELECT » classique sur une valeur string : on a souvent l'insensibilité à la casse, et dans cette nouvelle requête, il y aura la sensibilité à la casse). De même, c'est toujours conseillé de garder les mots de passe dans un DATATYPES « **VARBINARY** ».

```
CREATE TABLE IF NOT EXISTS categories (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(64) UNIQUE NOT NULL,
    description VARCHAR(255)
) ENGINE=InnoDB CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

- *Info importante pour les collations, les collations sensibles à la casse sur « utf8 » ou plus ne sont pas disponibles sur les versions précédentes de la 8.0.1.*

Les recherche FullText (MySQL)

Les recherches FullText permettent d'effectuer une **recherche dans la base de données de manière plus avancée** que l'opérateur LIKE et permettent aussi **d'organiser les résultats en fonction de la pertinence** (fonctionnant que sur les moteurs InnoDB et MyISAM sur des DATATYPES string). Pour l'utiliser, il faut tout simplement créer un nouvel index/option/colonne sur une table avec | FULLTEXT (colonne1, ...) ; |. Puis, on a trois façons d'utiliser le FullText dans le SELECT :

- | MATH (colonne1, ...) AGAINST ('recherche' IN NATURAL LANGUAGE MODE) ; | permet de donner des résultats selon leur pertinence. S'il y a de multiple mots, ce mode calcule le nombre de fois que chacun des mots séparément et ensemble sont présents.
- | MATH (colonne1, ...) AGAINST ('recherche' IN NATURAL LANGUAGE MODE WITH QUERY EXPENSION) ; | permet de faire presque pareil que le mode « NATURAL LANGUAGE » sauf que la recherche est plus large (prendre les erreurs d'inattention). Par contre, il faut faire attention pour ne pas avoir des résultats étonnants voire trop de résultats.
- | MATH (colonne1, ...) AGAINST ('recherche' IN BOOLEAN MODE) ; | permet de donner des résultats selon une recherche contrôlée car il faut soit ajouter des « + » sur certains mots (pour l'avoir dans la recherche), soit des « - » sur certains mots (pour ne pas l'avoir dans la recherche). De même, on peut ajouter un seul « + » sur un mot sur un ensemble de mots (pour avoir au moins un des mots dans la recherche => « OU »). Mais il peut avoir aussi d'autres opérateurs disponibles sur le site https://dev.mysql.com/doc/refman/8.0/en/fulltext-boolean.html. Par contre, ce mode n'organise pas la pertinence.

- *Le mode « NATURAL LANGUAGE » fait l'impasse sur certains mots du quotidien anglais dans la recherche. On les appelle les « STOPWORD ». Pour les connaître, il faut effectuer | SELECT * FROM INFORMATION_SHEMA.INNODB_FT_DEFAULT_STOPWORD ; |.*
- *Dans le mode « NATURAL LANGUAGE », on peut ajouter | '\"recherche\"' | pour que la recherche se concentre que sur les multiples mots ensembles.*
- *On peut avoir une notion de score quand on utilise dans la sélection (début, colonnes) du SELECT (la pertinence est négligée). Par exemple :*

```
SELECT *, MATCH (content) AGAINST ('raton laveur' IN BOOLEAN MODE) AS score /*Le score de la recherche
(de 0 à 1)*/
FROM information
WHERE MATCH (content) AGAINST ('raton laveur' IN BOOLEAN MODE); /*Ajouter la pertinence. Par contre, dans
le cas habituelle, il ne faut pas ajouter le WHERE*/
```

Les permissions (MySQL)

Un avantage de MySQL est sa capacité à pouvoir gérer facilement les niveaux d'accès à la base de données et aux tables grâce à un système d'administration. Par exemple pour donner l'accès à un utilisateur, on commence par créer son compte :

```
CREATE USER 'utilisateur'@'%' IDENTIFIED BY 'monmotdepasse';
```

- *Après le @, c'est le paramètre qui indique dans quel endroit peut-on se connecter. « % » indique qu'on peut se connecter n'importe tout.*
- *On peut aussi retrouver tous nos profils d'utilisateurs MySQL et leurs permissions avec | SELECT * FROM mysql.user ; |.*

Et puis, on va choisir quel privilège l'utilisateur a le droit avec *| GRANT privilège ON chemin_de_la_base_de_donnée TO utilisateur |*. On a énormément de privilèges qui sont disponibles sur MySQL (https://dev.mysql.com/doc/refman/5.7/en/grant.html). Je ne vais pas les énumérer car il me faudrait 5h à tous les comprendre. Donc les principaux accès spécifiques sont plus SELECT, ALL [PRIVILEGES], SHOW VIEW, SHOW TABLE, ALTER, CREATE, CREATE VIEW, DELETE, DROP, INDEX, INSERT, LOCK TABLE, TRIGGER, UPDATE et surtout USAGE (aucun privilège). Puis, on peut aussi supprimer un utilisateur avec *| DROP USER 'utilisateur'@'%' ; |*.

```
GRANT SELECT ON tutoriel.information TO 'utilisateur'@'%';
```

- *On peut aussi retrouver tous les privilèges d'un profil d'utilisateur avec | SHOW GRANTS FOR 'utilisateur'@'%' ; | et cela nous permet de supprimer les permissions avec | REVOKE permissionUtilisateur ; | (en commençant la permission par SELECT d'après le SHOW GRANT et en remplaçant TO par FROM).*

```
REVOKE SELECT ON `tutoriel`.`information` FROM 'utilisateur'@'%';
```

MySQLDump

Pour info, on a aussi un système **qui permet d'enregistrer les petites bases de données** comme un backup de secours se nommant MySQLDump. Par contre, il n'est pas disponible sur MAMP classique (⚠ Windows). Avec l'invite de commande (*|cd/d D:/|*), on peut sauvegarder, charger, en évitant les problèmes d'encodages => https://grafikart.fr/tutoriels/mysql-dump-2010.

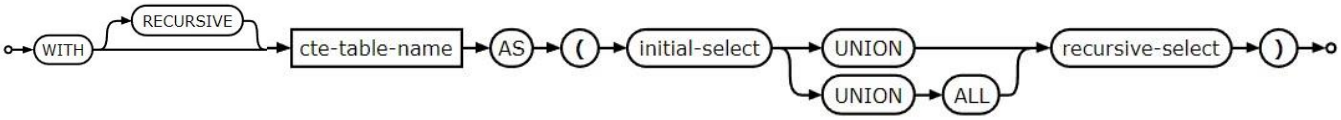
Bonus : les requêtes récursive (SQL)

Tout d'abord, qu'est-ce que la **récursivité**. C'est une manière de faire en algorithme où on rappelle la fonction parent dans son algorithme. Maintenant, que ça prend tout son sens, on peut l'utiliser pour de multiples choses à plusieurs cran/profondeur comme les arbres généalogique, une hiérarchie, … Et ba, si on n'utilise pas la récursivité sur certains problèmes, ça commence à être difficile avec les SELECT. Pour cela, nous allons voir comment écrire des requêtes pour récupérer des données récursives (recursive common table expressions). Le mot clef **WITH** permet d'écrire des déclarations auxiliaires que l'on pourra utiliser dans une requête plus large. Ces déclarations, souvent appelées Common Table Expression ou CTE, peuvent être vu comme des tables temporaires qui n'existe que pour une requête.

```
WITH
    cte1 AS (SELECT a, b FROM table1),
    cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

Sauf que cette méthode se sert de deux tables, non de manière récursive. Ainsi, on peut utiliser WITH afin de récupérer des données de manière récursive. La déclaration dans le AS sera décomposée en 2 déclarations regroupée par un UNION (ou UNION ALL) :

- ❖ Une première déclaration récupérera les données à la racine de notre récursion.
- ❖ Une seconde déclaration qui contiendra une référence au résultat de la requête précédente.



Pour l'exemple nous allons récupérer les catégories parentes récursivement :

```
WITH RECURSIVE temporytable AS (
    SELECT id, name, parent_id FROM categories WHERE id = 14 /* On récupère la catégorie en profondeur */
    UNION ALL
    SELECT c.id, c.name, c.parent_id FROM categories c, temporytable tt WHERE c.id = tt.parent_id
)
SELECT * FROM categories_tree
```

- *Chaque colonne et alias sélectionné dans une des deux déclaration doivent être ajoutés sur l'autre déclaration. De même, si on attribue des nom de colonnes (dans le bon ordre d'exécution) avant le « AS », les alias ne sont plus utiles.*

Ainsi (à l'inverse), on peut aussi récupérer les catégories enfants récursivement :

```
WITH RECURSIVE children (id, name, parent_id, level, path) AS (
    SELECT id, name, parent_id, 0, name FROM categories WHERE parent_id IS NULL
    UNION ALL
    SELECT
        c.id,
        c.name,
        c.parent_id,
        children.level + 1, -- Savoir combien d'enfants a cette espèce
        children.path || " > " || c.name --Le chemin généalogiques des enfants
    FROM categories c, children
    WHERE c.parent_id = children.id
)
SELECT * FROM children
```

- *Malheureusement, cette magnifique option est disponible que depuis les versions supérieures à MySQL 8. Autre information, le « WITH » reste trop gourmands en ressource pour les gros projets.*