Manipuler les différentes relations et opérations dans une BDD.

Modifier une table existante

→ ALTER TABLE

La commande `ALTER TABLE` permet de modifier la structure d'une table existante dans une base de données MySQL. Elle peut être utilisée pour ajouter, supprimer ou modifier des colonnes dans une table.

♦ Pour ajouter une colonne à une table, vous pouvez utiliser la syntaxe suivante:

```
ALTER TABLE nom_de_la_table ADD nom_de_la_colonne type_de_données;
```

Par exemple, pour ajouter une colonne `adresse` de type `VARCHAR` à une table `utilisateurs`, vous pouvez utiliser la commande suivante:

```
ALTER TABLE utilisateurs ADD adresse VARCHAR(255);
```

♦ Pour supprimer une colonne d'une table, vous pouvez utiliser la syntaxe suivante:

```
ALTER TABLE nom_de_la_table DROP COLUMN nom_de_la_colonne;
```

Par exemple, pour supprimer la colonne `adresse` de la table `utilisateurs`, vous pouvez utiliser la commande suivante:

```
ALTER TABLE utilisateurs DROP COLUMN adresse;
```

♦ Pour modifier une colonne existante dans une table, vous pouvez utiliser la syntaxe suivante:

```
ALTER TABLE nom_de_la_table MODIFY COLUMN nom_de_la_colonne nouveau_type_de_données;
```

Par exemple, pour modifier le type de données de la colonne `adresse` de la table `utilisateurs` en `TEXT`, vous pouvez utiliser la commande suivante:

```
ALTER TABLE utilisateurs MODIFY COLUMN adresse TEXT;
```

Il est important de noter que l'utilisation de la commande `ALTER TABLE` peut avoir des conséquences importantes sur les données stockées dans votre base de données. Il est

donc recommandé de bien réfléchir aux modifications que vous souhaitez effectuer avant d'utiliser cette commande.

Relation un à plusieurs

Il s'agit d'une relation entre deux tables dans une base de données relationnelle où une ligne d'une table (la table "un") peut être liée à plusieurs lignes de l'autre table (la table "plusieurs"). Par exemple, si vous avez une table `auteurs` et une table `livres`, une relation un à plusieurs entre ces deux tables signifie qu'un auteur peut avoir écrit plusieurs livres, mais qu'un livre ne peut avoir qu'un seul auteur.

Pour créer une relation un à plusieurs entre deux tables dans MySQL, vous pouvez utiliser une clé étrangère. C'est une colonne dans la table "plusieurs" qui fait référence à la clé primaire de la table "un". Cette clé étrangère crée un lien entre les deux tables et permet de s'assurer que les données sont cohérentes.

Voici un exemple de création d'une relation un à plusieurs entre une table `auteurs` et une table `livres`:

```
CREATE TABLE auteurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL
);

CREATE TABLE livres (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titre VARCHAR(255) NOT NULL,
    auteur_id INT,
    FOREIGN KEY (auteur_id) REFERENCES auteurs(id)
);
```

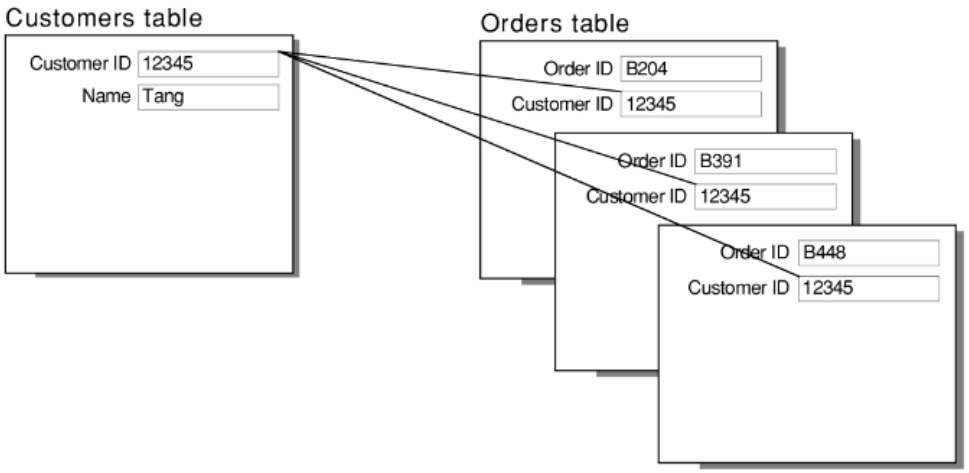
Dans cet exemple, nous avons créé deux tables: `auteurs` et `livres`. La table `auteurs` à une colonne `id` qui est la clé primaire de la table. La table `livres` a une colonne `auteur_id` qui est une clé étrangère faisant référence à la colonne `id` de la table `auteurs`. Cette clé étrangère crée une relation un à plusieurs entre les deux tables : ici, un livre ne peut avoir qu'un seul auteur, mais un auteur peut avoir écrit plusieurs livres.

Pour insérer des données dans ces deux tables et créer des liens entre elles, vous pouvez utiliser des requêtes `INSERT` pour insérer des données dans les deux tables, en spécifiant la valeur de la clé étrangère pour lier les lignes des deux tables. Par exemple, pour insérer un auteur et deux livres écrits par cet auteur, vous pouvez utiliser les requêtes suivantes:

```
INSERT INTO auteurs (nom) VALUES ('J.K. Rowling');

INSERT INTO livres (titre, auteur_id) VALUES ('Harry Potter à l\'école des sorciers', 1);
INSERT INTO livres (titre, auteur_id) VALUES ('Harry Potter et la Chambre des Secrets', 1);
```

Ces requêtes insèrent un nouvel auteur dans la table `auteurs`, puis insèrent deux nouveaux livres dans la table `livres`, en spécifiant la valeur de la colonne `auteur_id` pour lier ces livres à l'auteur que nous venons d'insérer.

Voici un schéma pour illustrer une relation un à plusieurs (one-to-many) avec MySQL :



Dans cet exemple, nous avons deux tables: `Auteurs` et `Livres`. La table `Auteurs` à une colonne `id` qui est la clé primaire de la table. La table `Livres` a une colonne `auteur_id` qui est une clé étrangère faisant référence à la colonne `id` de la table `Auteurs`. Cette clé étrangère crée une relation un à plusieurs entre les deux tables: un livre ne peut avoir qu'un seul auteur, mais un auteur peut avoir écrit plusieurs livres.

Résumé

Pour créer une relation un à plusieurs (one-to-many) dans une base de données MySQL, vous pouvez suivre les étapes suivantes:

1. Créez les deux tables qui seront impliquées dans la relation. L'une des tables représentera l'entité "un" et l'autre représentera l'entité "plusieurs".
2. Ajoutez une clé étrangère à la table "plusieurs" qui fera référence à la clé primaire de la table "un". Cette clé étrangère crée un lien entre les deux tables et permet de s'assurer que les données sont cohérentes.
3. Utilisez la commande `ADD FOREIGN KEY` pour définir une contrainte de clé étrangère entre la clé étrangère de la table "plusieurs" et la clé primaire de la table "un". Cette contrainte garantit que les valeurs de la clé étrangère dans la table "plusieurs" correspondent à des valeurs existantes dans la clé primaire de la table "un".
4. Insérez des données dans les deux tables en spécifiant les valeurs appropriées pour la clé étrangère dans la table "plusieurs" pour lier les lignes des deux tables.

Relation plusieurs à plusieurs (many to many)

C'est une relation entre deux tables dans une base de données relationnelle où plusieurs lignes d'une table peuvent être liées à plusieurs lignes de l'autre table.

Par exemple, si vous avez une table `étudiants` et une table `cours`, une relation plusieurs à plusieurs entre ces deux tables signifie qu'un étudiant peut être inscrit à plusieurs cours et qu'un cours peut avoir plusieurs étudiants inscrits.

Pour créer une relation plusieurs à plusieurs entre deux tables dans MySQL, vous devez utiliser une table intermédiaire (également appelée table de jonction ou table associative) pour stocker les associations entre les deux tables. Cette table intermédiaire contient des clés étrangères faisant référence aux clés primaires des deux tables impliquées dans la relation.

Voici un exemple de création d'une relation plusieurs à plusieurs entre une table `étudiants` et une table `cours`:

```
CREATE TABLE étudiants (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL
);

CREATE TABLE cours (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titre VARCHAR(255) NOT NULL
);

CREATE TABLE inscriptions (
    étudiant_id INT,
    cours_id INT,
    FOREIGN KEY (étudiant_id) REFERENCES étudiants(id),
    FOREIGN KEY (cours_id) REFERENCES cours(id)
);
```

Dans cet exemple, nous avons créé trois tables: `étudiants`, `cours` et `inscriptions`. La table `étudiants` a une colonne `id` qui est la clé primaire de la table. La table `cours` a également une colonne `id` qui est la clé primaire de la table. La table `inscriptions` est la table intermédiaire qui stocke les associations entre les étudiants et les cours. Cette table a deux colonnes: `étudiant_id` et `cours_id`, qui sont des clés étrangères faisant référence aux colonnes `id` des tables `étudiants` et `cours`, respectivement.

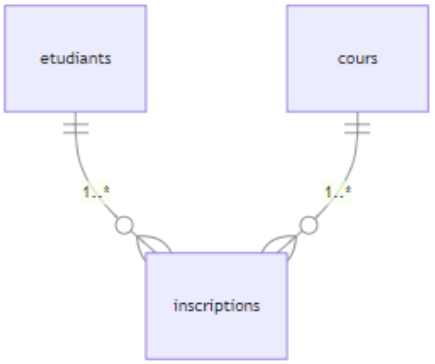
Pour insérer des données dans ces trois tables et créer des liens entre elles, vous pouvez utiliser des requêtes `INSERT` pour insérer des données dans les tables `étudiants` et `cours`, puis utiliser des requêtes `INSERT` pour insérer des données dans la table `inscriptions`, en spécifiant les valeurs des clés étrangères pour lier les lignes des deux autres tables. Par exemple, pour insérer un étudiant, un cours et une inscription liant cet étudiant à ce cours, vous pouvez utiliser les requêtes suivantes:

```
INSERT INTO étudiants (nom) VALUES ('Alice');

INSERT INTO cours (titre) VALUES ('Mathématiques');

INSERT INTO inscriptions (étudiant_id, cours_id) VALUES (1, 1);
```

Ces requêtes insèrent un nouvel étudiant dans la table `étudiants`, un nouveau cours dans la table `cours`, puis une nouvelle inscription dans la table `inscriptions`, en spécifiant les valeurs des colonnes `étudiant_id` et `cours_id` pour lier l'étudiant et le cours que nous venons d'insérer.

Schéma pour illustrer une relation plusieurs à plusieurs (many-to-many) avec MySQL:

```
+------------+     +-----------------+     +------------+
|  Étudiants |     |  Inscriptions   |     |   Cours    |
+------------+     +-----------------+     +------------+
| # id       |     | - étudiant_id   |     | # id       |
| - nom      |-----| - cours_id      |-----| - titre    |
+------------+     +-----------------+     +------------+
```



Dans ce schéma, nous avons trois entités: `Étudiants`, `Inscriptions` et `Cours`. L'entité `Étudiants` a un attribut `id` qui est la clé primaire de l'entité. L'entité `Cours` a également un attribut `id` qui est la clé primaire de l'entité. L'entité `Inscriptions` est l'entité intermédiaire qui stocke les associations entre les étudiants et les cours. Cette entité a deux attributs: `étudiant_id` et `cours_id`, qui sont des clés étrangères faisant référence aux attributs `id` des entités `Étudiants` et `Cours`, respectivement.

Le symbole `#` devant les attributs `id` indique que ces attributs sont des clés primaires. Les symboles `-` devant les autres attributs indiquent que ce sont des attributs ordinaires. Les lignes entre les entités indiquent qu'il existe une relation entre elles, et les symboles `-----` indiquent que cette relation est de type plusieurs à plusieurs.

Résumé

Pour créer une relation plusieurs à plusieurs (many-to-many) dans une base de données MySQL, vous pouvez suivre les étapes suivantes:

1. Créez les deux tables qui seront impliquées dans la relation. Ces deux tables représenteront les entités qui ont une relation plusieurs à plusieurs entre elles.
2. Créez une table intermédiaire (également appelée table de jonction ou table associative) pour stocker les associations entre les deux tables. Cette table intermédiaire contiendra des clés étrangères faisant référence aux clés primaires des deux tables impliquées dans la relation.
3. Utilisez la commande `ADD FOREIGN KEY` pour définir des contraintes de clé étrangère entre les clés étrangères de la table intermédiaire et les clés primaires des deux autres tables. Ces contraintes garantissent que les valeurs des clés étrangères dans la table intermédiaire correspondent à des valeurs existantes dans les clés primaires des deux autres tables.
4. Insérez des données dans les trois tables en spécifiant les valeurs appropriées pour les clés étrangères dans la table intermédiaire pour lier les lignes des deux autres tables.

Relation un à un (one-to-one)

C'est une relation entre deux tables dans une base de données relationnelle où une ligne d'une table ne peut être liée qu'à une seule ligne de l'autre table. Par exemple, si vous avez une table `personnes` et une table `permis_de_conduire`, une relation un à un entre ces deux tables signifie qu'une personne ne peut avoir qu'un seul permis de conduire et qu'un permis de conduire ne peut être associé qu'à une seule personne.

Pour créer une relation un à un entre deux tables dans MySQL, vous pouvez utiliser une clé étrangère, qui sera une colonne dans l'une des tables à faire référence à la clé primaire de l'autre table. Cette clé étrangère crée un lien entre les deux tables et permet de s'assurer que les données sont cohérentes.

Voici un exemple de création d'une relation un à un entre une table `personnes` et une table `permis_de_conduire`:

```
CREATE TABLE personnes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL
);

CREATE TABLE permis_de_conduire (
    id INT AUTO_INCREMENT PRIMARY KEY,
    numero VARCHAR(255) NOT NULL,
    personne_id INT,
    FOREIGN KEY (personne_id) REFERENCES personnes(id)
);
```

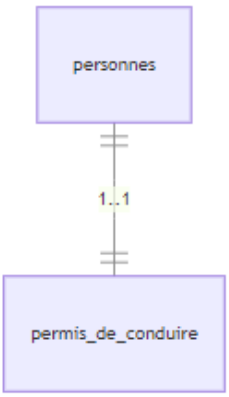
Dans cet exemple, nous avons créé deux tables: `personnes` et `permis_de_conduire`. La table `personnes` a une colonne `id` qui est la clé primaire de la table. La table `permis_de_conduire` a une colonne `personne_id` qui est une clé étrangère faisant référence à la colonne `id` de la table `personnes`. Cette clé étrangère crée une relation un à un entre les deux tables: un permis de conduire ne peut être associé qu'à une seule personne, et une personne ne peut avoir qu'un seul permis de conduire.

Pour insérer des données dans ces deux tables et créer des liens entre elles, vous pouvez utiliser des requêtes `INSERT` pour insérer des données dans les deux tables, en spécifiant la valeur de la clé étrangère pour lier les lignes des deux tables. Par exemple, pour insérer une personne et son permis de conduire, vous pouvez utiliser les requêtes suivantes:

```
INSERT INTO personnes (nom) VALUES ('Alice');

INSERT INTO permis_de_conduire (numero, personne_id) VALUES ('123456', 1);
```

Ces requêtes insèrent une nouvelle personne dans la table `personnes`, puis insèrent un nouveau permis de conduire dans la table `permis_de_conduire`, en spécifiant la valeur de la colonne `personne_id` pour lier le permis de conduire à la personne que nous venons d'insérer.

Schéma pour illustrer une relation un à un (one-to-one) avec MySQL:



Dans ce schéma, nous avons deux entités: `Personnes` et `Permis_de_conduire`. L'entité `Personnes` a un attribut `id` qui est la clé primaire de l'entité. L'entité `Permis_de_conduire` a également un attribut `id` qui est la clé primaire de l'entité, ainsi qu'un attribut `personne_id` qui est une clé étrangère faisant référence à l'attribut `id` de l'entité `Personnes`. Cette clé étrangère crée une relation un à un entre les deux entités: un permis de conduire ne peut être associé qu'à une seule personne, et une personne ne peut avoir qu'un seul permis de conduire.

Le symbole `#` devant les attributs `id` indique que ces attributs sont des clés primaires. Le symbole `-` devant les autres attributs indique que ce sont des attributs ordinaires. La ligne entre les entités indique qu'il existe une relation entre elles, et le symbole `-----` indique que cette relation est de type un à un.

Résumé

Pour créer une relation un à un (one-to-one) dans une base de données MySQL, vous pouvez suivre les étapes suivantes:

1. Créez les deux tables qui seront impliquées dans la relation. Ces deux tables représenteront les entités qui ont une relation un à un entre elles.
2. Ajoutez une clé étrangère à l'une des tables pour faire référence à la clé primaire de l'autre table. Cette clé étrangère crée un lien entre les deux tables et permet de s'assurer que les données sont cohérentes.
3. Utilisez la commande `ADD FOREIGN KEY` pour définir une contrainte de clé étrangère entre la clé étrangère et la clé primaire de l'autre table. Cette contrainte garantit que les valeurs de la clé étrangère correspondent à des valeurs existantes dans la clé primaire de l'autre table.
4. Insérez des données dans les deux tables en spécifiant les valeurs appropriées pour la clé étrangère pour lier les lignes des deux tables.

Table récursive

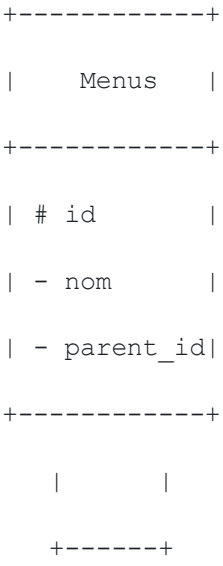
Exemple de table récursive pour la création d'un menu et de ses sous-menus dans une base de données MySQL:

```sql
CREATE TABLE menus (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(255) NOT NULL,
    parent_id INT,
    FOREIGN KEY (parent_id) REFERENCES menus(id)
);

INSERT INTO menus (nom, parent_id) VALUES ('Accueil', NULL);
INSERT INTO menus (nom, parent_id) VALUES ('Produits', NULL);
INSERT INTO menus (nom, parent_id) VALUES ('Services', NULL);
INSERT INTO menus (nom, parent_id) VALUES ('Contact', NULL);

INSERT INTO menus (nom, parent_id) VALUES ('Produit 1', 2);
INSERT INTO menus (nom, parent_id) VALUES ('Produit 2', 2);
INSERT INTO menus (nom, parent_id) VALUES ('Produit 3', 2);

INSERT INTO menus (nom, parent_id) VALUES ('Service 1', 3);
INSERT INTO menus (nom, parent_id) VALUES ('Service 2', 3);
INSERT INTO menus (nom, parent_id) VALUES ('Service 3', 3);
```

Dans cet exemple, nous avons créé une table `menus` avec les colonnes `id`, `nom` et `parent_id`. La colonne `id` est la clé primaire de la table. La colonne `parent_id` est une clé étrangère faisant référence à la colonne `id` de la même table. Cette clé étrangère crée une relation récursive entre les lignes de la table: un menu peut avoir un menu parent et plusieurs menus enfants.

Des données ont ensuite été insérées dans cette table pour créer un menu avec plusieurs sous-menus. Les quatre premières lignes insérées représentent les éléments du menu principal, avec des valeurs `NULL` pour la colonne `parent_id` pour indiquer qu'ils n'ont pas de menu parent. Les lignes suivantes représentent les sous-menus des éléments "Produits" et "Services", avec des valeurs pour la colonne `parent_id` pour indiquer à quel élément du menu principal ils sont liés.

Schéma pour illustrer l'exemple de table récursive pour la création d'un menu et de ses sous-menus dans une base de données MySQL:



Dans ce schéma, nous avons une entité `Menus` avec trois attributs: `id`, `nom` et `parent_id`. L'attribut `id` est la clé primaire de l'entité. L'attribut `parent_id` est une clé étrangère faisant référence à l'attribut `id` de la même entité. Cette clé étrangère crée une relation récursive entre les lignes de la table: un menu peut avoir un menu parent et plusieurs menus enfants.

Le symbole `#` devant l'attribut `id` indique que cet attribut est une clé primaire. Le symbole `-` devant les autres attributs indique que ce sont des attributs ordinaires. La ligne entre l'entité et elle-même indique qu'il existe une relation récursive, et le symbole `+` indique que cette relation est de type un à plusieurs.

Résumé

Pour créer une table récursive dans une base de données MySQL, vous pouvez suivre les étapes suivantes:

1. Créez une table avec les colonnes appropriées pour stocker les données que vous souhaitez représenter. Cette table représentera l'entité qui a une relation récursive avec elle-même.
2. Ajoutez une clé étrangère à la table pour faire référence à sa propre clé primaire. Cette clé étrangère crée un lien récursif entre les lignes de la table.
3. Utilisez la commande `ADD FOREIGN KEY` pour définir une contrainte de clé étrangère entre la clé étrangère et la clé primaire de la table. Cette contrainte garantit que les valeurs de la clé étrangère correspondent à des valeurs existantes dans la clé primaire de la table.
4. Insérez des données dans la table en spécifiant les valeurs appropriées pour la clé étrangère pour lier les lignes de la table entre elles.

Les jointures

Elles permettent de combiner des données provenant de plusieurs tables en une seule requête. Il existe plusieurs types de jointures, chacun ayant un comportement différent en termes de gestion des données qui ne correspondent pas dans les tables jointes.

Les types de jointures les plus couramment utilisés en MySQL sont les suivants:

- **INNER JOIN** : Cette jointure renvoie uniquement les lignes qui ont des valeurs correspondantes dans les deux tables. C'est le type de jointure le plus couramment utilisé.

- **LEFT JOIN** : Cette jointure renvoie toutes les lignes de la table de gauche, ainsi que les lignes correspondantes de la table de droite. Si aucune ligne ne correspond dans la table de droite, les valeurs des colonnes de cette table seront `NULL`.

⏵ **RIGHT JOIN** : Cette jointure renvoie toutes les lignes de la table de droite, ainsi que les lignes correspondantes de la table de gauche. Si aucune ligne ne correspond dans la table de gauche, les valeurs des colonnes de cette table seront `NULL`.

⏵ **FULL OUTER JOIN** : Cette jointure renvoie toutes les lignes des deux tables, avec les lignes correspondantes dans les deux tables. Si aucune ligne ne correspond dans l'une ou l'autre des tables, les valeurs des colonnes de cette table seront `NULL`.

⏵ **CROSS JOIN** : Cette jointure renvoie le produit cartésien des deux tables, c'est-à-dire toutes les combinaisons possibles entre les lignes des deux tables.

Voici un exemple d'utilisation d'une jointure **INNER JOIN** pour combiner des données provenant des tables "Auteurs" et "Livres" :

```
SELECT Auteurs.nom, Livres.titre
FROM Auteurs
INNER JOIN Livres
ON Auteurs.id = Livres.auteur_id;
```

Dans cet exemple, la requête renverra le nom des auteurs et le titre des livres qu'ils ont écrits. Seuls les auteurs ayant écrit au moins un livre apparaîtront dans les résultats.

Que va faire cette requête avec la base aliments?

SELECT aliments.nom, aliments.id, legumes.nom, legumes.id
FROM aliments
INNER JOIN legumes
ON aliments.id = legumes.id;

→ Cette requête renvoie les recettes et leurs aliments associés, mais seulement si une association existe dans la table `est_composé_de` INNER JOIN :

```
SELECT r.name_recette, a.nom_aliment
FROM recette r
INNER JOIN est_composé_de e
ON r.id_recette = e.id_recette
INNER JOIN aliment a
ON e.id_aliment = a.id_aliment;
```

Cette requête renvoie toutes les recettes, même celles qui n'ont pas d'aliments associés. Si une recette n'a pas d'aliments associés, les colonnes correspondantes à l'aliment seront NULL.

LEFT JOIN :

```
SELECT r.name_recette, a.nom_aliment
FROM recette r
LEFT JOIN est_composé_de e
ON r.id_recette = e.id_recette
LEFT JOIN aliment a
ON e.id_aliment = a.id_aliment;
```

Cette requête renvoie tous les aliments, même ceux qui ne sont associés à aucune recette. Si un aliment n'est associé à aucune recette, les colonnes correspondantes à la recette seront NULL. (Notez que `RIGHT JOIN` est moins couramment utilisé que `LEFT JOIN`, et vous pourriez obtenir le même résultat en inversant les tables et en utilisant un `LEFT JOIN`).

RIGHT JOIN :

```
SELECT r.name_recette, a.nom_aliment
FROM recette r
RIGHT JOIN est_composé_de e
ON r.id_recette = e.id_recette
RIGHT JOIN aliment a
ON e.id_aliment = a.id_aliment;
```

Ces requêtes vous permettent de voir comment les différentes jointures fonctionnent et comment elles peuvent être utilisées pour récupérer des données de tables liées.

Quand utiliser les jointures ?

Les jointures sont un outil puissant pour combiner des données provenant de plusieurs tables en une seule requête. Elles sont particulièrement utiles lorsque les données sont stockées dans des tables séparées mais liées, et que l'on souhaite récupérer des informations à partir de ces différentes tables en une seule requête

Procédures Stockées avec la base `fruits_legumes`

Il s'agit d'un groupe de commandes SQL qui peuvent être exécutées en une seule fois. Elles sont utilisées pour encapsuler une logique métier, pour automatiser des processus ou pour regrouper des commandes SQL qui sont exécutées ensemble.

Les avantages des Procédures Stockées

1. **Réutilisabilité** : Une fois créée, la procédure stockée peut être appelée plusieurs fois.
2. **Maintenance** : La logique métier est centralisée. Si besoin de modification, il suffit de mettre à jour la procédure stockée.

3. **Performance** : Les procédures stockées sont compilées et leur plan d'exécution est sauvegardé, ce qui peut accélérer leur exécution.
4. **Sécurité** : Vous pouvez accorder des droits spécifiques sur une procédure stockée sans exposer la table sous-jacente.

→ Création d'une Procédure Stockée avec `fruits_legumes`

Supposons que nous voulons créer une procédure stockée pour obtenir tous les fruits et légumes de couleur spécifique

```
DELIMITER // CREATE PROCEDURE GetProduitsByColor(IN col VARCHAR(255)) BEGIN SELECT 'Fruit' AS Type, nom FROM fruits_legumes.fruits WHERE couleur = col; UNION SELECT 'Légume' AS Type, nom FROM fruits_legumes.legumes WHERE couleur = col; END // DELIMITER ;
```

→ Appel d'une Procédure Stockée

Pour appeler la procédure stockée que nous venons de créer :

```
CALL GetProduitsByColor('Vert');
```

→ Modification d'une Procédure Stockée

Si vous devez apporter des modifications à votre procédure stockée :

```
DELIMITER // ALTER PROCEDURE GetProduitsByColor(IN col VARCHAR(255)) BEGIN -- (Votre nouvelle logique ici) END // DELIMITER ;
```

→ Suppression d'une Procédure Stockée

Pour supprimer une procédure stockée :

```
DROP PROCEDURE IF EXISTS GetProduitsByColor;
```

Bonnes Pratiques

1. **Documentation** : Commentez toujours votre code pour expliquer la logique métier.
2. **Simplicité** : Essayez de garder les procédures stockées simples et focalisées. Si une procédure devient trop complexe, envisagez de la diviser.
3. **Sécurité** : Veillez à ne pas exposer d'informations sensibles via vos procédures stockées.

Résumé

Les procédures stockées offrent une manière puissante de regrouper et de centraliser la logique métier. Bien utilisées, elles peuvent améliorer la réutilisabilité, la maintenance, la performance et la sécurité de vos applications de base de données.

Sécurisation des Bases de Données

C'est primordial dans le monde informatique actuel. Une faille dans votre base de données peut entraîner des pertes financières, une atteinte à la réputation et des violations de la vie privée.

1. Contrôle d'accès

Objectif: Limiter l'accès à la base de données à des utilisateurs autorisés.

- Mots de passe forts: Assurez-vous que tous les utilisateurs ont des mots de passe robustes.

```
ALTER USER 'username'@'localhost' IDENTIFIED BY 'StrongPassword!';
```

- Privilèges limités: N'accordez que les privilèges nécessaires. Si un utilisateur n'a pas besoin d'écrire dans la base, ne lui donnez que des droits de lecture.

```
GRANT SELECT ON fruits_legumes.* TO 'username'@'localhost';
```

2. Protection contre les injections SQL

Objectif: Empêcher les attaquants d'exécuter des requêtes arbitraires.

Utiliser des requêtes préparées: Cela garantit que les données transmises à la base de données sont traitées comme des données et non comme du code.
Exemple en PHP :

```
$stmt = $conn->prepare("INSERT INTO fruits (nom, couleur) VALUES (?, ?)"); $stmt->bind_param("ss", $nom, $couleur);
```

PHP (en utilisant PDO) :

```php
<?php
$host = 'localhost';
$db   = 'fruits_legumes';
$user = 'username';
$pass = 'password';
$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];

try {
    $pdo = new PDO($dsn, $user, $pass, $options);
    $stmt = $pdo->prepare("INSERT INTO fruits (nom, couleur) VALUES (?, ?)");
    $stmt->execute(['Pomme', 'Rouge']);
    echo $stmt->rowCount()." rows inserted.";
} catch (PDOException $e) {
    throw new PDOException($e->getMessage(), (int)$e->getCode());
}
?>
```

Java (en utilisant JDBC) :

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class Main {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/fruits_legumes";
        String user = "username";
        String password = "password";

        try {
            Connection conn = DriverManager.getConnection(url, user, password);

            String query = "INSERT INTO fruits (nom, couleur) VALUES (?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setString(1, "Pomme");
            pstmt.setString(2, "Rouge");

            int rowsAffected = pstmt.executeUpdate();
            System.out.println(rowsAffected + " row(s) inserted.");

            pstmt.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Node.js (en utilisant `mysql` npm package) :

```javascript
const mysql = require('mysql');

const connection = mysql.createConnection({
    host: 'localhost',
    user: 'username',
    password: 'password',
    database: 'fruits_legumes'
});

connection.connect();

const fruitName = 'Pomme';
const fruitColor = 'Rouge';
const query = 'INSERT INTO fruits (nom, couleur) VALUES (?, ?)';

connection.query(query, [fruitName, fruitColor], (error, results, fields) => {
    if (error) throw error;
    console.log(results.affectedRows + ' row(s) inserted.');
});

connection.end();
```

C# (en utilisant ADO.NET) :

```
using System;
using System.Data.SqlClient;

class Program {
    static void Main() {
        string connectionString = @"Data Source=localhost;Initial Catalog=fruits_legumes;User Id=username;Password=password;";
        using (SqlConnection connection = new SqlConnection(connectionString)) {
            connection.Open();

            string query = "INSERT INTO fruits (nom, couleur) VALUES (@name, @color)";
            using (SqlCommand command = new SqlCommand(query, connection)) {
                command.Parameters.AddWithValue("@name", "Pomme");
                command.Parameters.AddWithValue("@color", "Rouge");

                int rowsAffected = command.ExecuteNonQuery();
                Console.WriteLine(rowsAffected + " row(s) inserted.");
            }

            connection.Close();
        }
    }
}
```

Python (en utilisant `mysql-connector-python`) :

```
import mysql.connector

connection = mysql.connector.connect(
    host="localhost",
    user="username",
    password="password",
    database="fruits_legumes"
)

cursor = connection.cursor()
query = "INSERT INTO fruits (nom, couleur) VALUES (%s, %s)"
data = ("Pomme", "Rouge")

cursor.execute(query, data)
connection.commit()
print(cursor.rowcount, "Record inserted.")

cursor.close()
connection.close()
```

Rust (en utilisant `mysql` crate) :

```rust
extern crate mysql;

use mysql as my;

fn main() {
    let pool = my::Pool::new("mysql://username:password@localhost:3306/fruits_legumes").unwrap();

    let mut stmt = pool.prepare("INSERT INTO fruits (nom, couleur) VALUES (?, ?)").unwrap();

    stmt.execute(("Pomme", "Rouge")).unwrap();

    println!("Row inserted.");
}
```

- Pour Rust, vous aurez besoin d'ajouter `mysql = "X.X.X"` (où "X.X.X" est la version actuelle) à votre fichier `Cargo.toml` pour inclure le “crate mysql”.

3. Mise à jour régulière

Objectif: Protéger votre base de données des vulnérabilités connues.

Gardez votre système de gestion de base de données (SGBD) à jour. Les mises à jour incluent souvent des correctifs de sécurité.

4. Sauvegarde régulière

Objectif: Assurer une récupération rapide en cas de perte de données.

- Effectuez des sauvegardes régulières de votre base de données.
  ```
  mysqldump -u [username] -p fruits_legumes > backup.sql
  ```

5. Sécurisation du réseau

Objectif: Limiter et surveiller l'accès au serveur de base de données.

- Utilisez un pare-feu pour limiter l'accès à votre SGBD.
- Si possible, utilisez des connexions chiffrées, comme SSL, pour protéger les données en transit.

6. Audits et Logs

Objectif: Surveiller et examiner l'activité de la base de données.

- Activez la journalisation pour suivre les accès et modifications.
- Examinez régulièrement les logs pour détecter toute activité suspecte.

7. Minimisation des données

Objectif: Réduire le risque en limitant les données stockées.

- Ne stockez que les données nécessaires. Si vous n'avez pas besoin d'une information spécifique, ne la collectez pas et ne la stockez pas.

Résumé

La sécurité des bases de données est un processus continu qui nécessite une vigilance constante. Avec les bonnes pratiques et une approche proactive, vous pouvez grandement réduire les risques associés à votre base de données.

ENGINE Moteur de stockage

En MySQL, il existe plusieurs moteurs de stockage (ou "engines") disponibles en plus de `InnoDB`. Voici quelques-uns des moteurs de stockage les plus couramment utilisés :

1. **MyISAM** :
   - C'était le moteur de stockage par défaut avant que `InnoDB` ne le remplace.
   - Ne supporte pas les transactions.
   - Supporte le verrouillage au niveau de la table plutôt qu'au niveau de la ligne.
   - Généralement plus rapide pour les opérations de lecture, mais peut être plus lent pour les opérations d'écriture en raison du verrouillage au niveau de la table.
2. **MEMORY (ou HEAP)** :
   - Stocke toutes ses données en mémoire, ce qui le rend très rapide.
   - Les données sont perdues si le serveur est arrêté ou redémarré.
   - Utile pour les tables temporaires ou les caches.
3. **CSV** :
   - Stocke les données sous forme de fichiers CSV.
   - Ne supporte pas les index.
4. **ARCHIVE** :
   - Utilisé pour stocker de grandes quantités de données sans index.
   - Bon pour l'archivage des données.
5. **BLACKHOLE** :
   - Accepte les données mais ne les stocke pas (les données sont immédiatement jetées).
   - Utile dans des scénarios de réplication où vous ne voulez pas stocker de données sur un serveur esclave.
6. **FEDERATED** :
   - Permet d'accéder à des données stockées sur un autre serveur MySQL.
   - N'est pas inclus par défaut dans les installations MySQL récentes.
7. **MERGE** :
   - Regroupe plusieurs tables MyISAM en une seule table.
8. **NDB (ou NDBCLUSTER)** :
   - Moteur de stockage pour MySQL Cluster.

- Stockage distribué, haute disponibilité.

Il est important de noter que chaque moteur de stockage a ses propres avantages, inconvénients et cas d'utilisation spécifiques. `InnoDB` est actuellement le moteur de stockage par défaut pour MySQL en raison de ses caractéristiques telles que le support des transactions, la cohérence des données et le verrouillage au niveau de la ligne.