



- 
- Base de données relationnelle
  - Jointure Interne
  - Jointure externe
  - Union
  - Coalesce
  - Jointure suite
  - Jointure naturelle
  - Jointure auto
  - Cross Join
  - Les commentaires

Une base de données relationnelle en SQL (Structured Query Language) est un type de base de données qui utilise des relations entre les données pour organiser et stocker les informations. Les données sont organisées en tables, où chaque table représente une entité et chaque colonne représente un attribut de cette entité.

Les tables peuvent être reliées entre elles par des clés étrangères, permettant de créer des relations entre les différentes entités. Les requêtes SQL peuvent être utilisées pour récupérer des données à partir de ces tables et pour effectuer des opérations telles que la sélection, la modification, l'insertion et la suppression de données.

Les bases de données relationnelles sont utilisées dans de nombreuses applications, telles que les systèmes de gestion de contenu, les systèmes de gestion de la relation client (CRM), les systèmes de gestion des ressources humaines et de nombreuses autres applications qui nécessitent une gestion et une manipulation efficaces de grandes quantités de données.

Les relations entre les tables dans une base de données relationnelle sont créées à l'aide de clés étrangères. Les clés étrangères sont des colonnes dans une table qui référencent les valeurs de clé primaire d'une autre table.

La clé primaire d'une table est une colonne ou un ensemble de colonnes qui identifie de manière unique chaque enregistrement dans cette table. Les clés étrangères sont utilisées pour établir des relations entre les tables et permettent de naviguer entre les données.

Il existe trois types de relations possibles entre les tables :

- Une relation "un à un" : chaque enregistrement dans une table ne peut être lié qu'à un seul enregistrement dans l'autre table.
- Une relation "un à plusieurs" : chaque enregistrement dans une table peut être lié à plusieurs enregistrements dans l'autre table.
- Une relation "plusieurs à plusieurs" : plusieurs enregistrements dans une table peuvent être liés à plusieurs enregistrements dans l'autre table.

Pour illustrer cela, prenons l'exemple d'une base de données pour une entreprise qui gère des projets et des employés. Nous avons deux tables : une table "employés" avec une clé primaire "ID Employé" et une table "projets" avec une clé primaire "ID Projet". Nous pouvons créer une relation "plusieurs à plusieurs" entre ces deux tables en ajoutant une troisième table "affectations" qui contiendra des clés étrangères "ID Employé" et "ID Projet", ainsi que d'autres informations comme la date de début et la date de fin de l'affectation.

Ainsi, chaque enregistrement dans la table "employés" peut être lié à plusieurs enregistrements dans la table "projets" via la table "affectations", et chaque enregistrement dans la table "projets" peut être lié à plusieurs enregistrements dans la table "employés" via cette même table "affectations". Cela permet de représenter efficacement les affectations des employés à différents projets.

En SQL, une clé primaire (primary key) est une colonne ou un ensemble de colonnes qui identifient de manière unique chaque ligne d'une table. Chaque table ne peut avoir qu'une seule clé primaire. La clé primaire garantit l'intégrité des données dans la table en empêchant les doublons et en permettant aux autres tables de se référer à des enregistrements spécifiques.

Une clé étrangère (foreign key) est une colonne ou un ensemble de colonnes qui fait référence à la clé primaire d'une autre table. Elle permet de lier les données de deux tables différentes. Les clés étrangères garantissent l'intégrité référentielle en empêchant l'insertion de données qui ne sont pas présentes dans la table parente.

Par exemple, si nous avons une table "client" avec une clé primaire "client\_id", et une autre table "commande" avec une clé primaire "com\_id" et une clé étrangère "client\_id" qui fait référence à la clé primaire "client\_id" de la table "client", cela signifie qu'une commande ne peut être créée que pour un client existant dans la table "clients".

En résumé, la clé primaire est utilisée pour identifier de manière unique chaque ligne d'une table, tandis que la clé étrangère est utilisée pour lier les données de deux tables différentes. Les clés primaires et étrangères sont essentielles pour garantir l'intégrité des données dans une base de données relationnelle.



## Créer un index:

En créant un index sur les colonnes utilisées dans la requête, MySQL peut trouver rapidement les résultats correspondants sans avoir à parcourir toutes les lignes de la table. Cela peut considérablement améliorer les performances de votre application.

La syntaxe est la suivante:

```
CREATE INDEX nom_index ON ma_table  
(colonne1, colonne2);
```

## Créer des relations via l'onglet concepteur sur PHPmyAdmin

Lorsque vous créez une relation entre deux tables dans MySQL (via PhpMyAdmin ou toute autre interface), vous pouvez définir les options de suppression ("ON DELETE") et de mise à jour ("ON UPDATE") pour la clé étrangère.

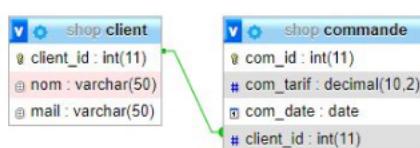
Voici ce que ces options signifient :

- **ON DELETE** : Cette option détermine ce qui se passe dans la table référencée (la table à laquelle la clé étrangère fait référence) lorsqu'une ligne est supprimée de la table parente (la table qui contient la clé étrangère). Les options courantes sont :
  - **CASCADE** : Supprime automatiquement toutes les lignes de la table référencée qui sont liées à la ligne supprimée de la table parente.
  - **SET NULL** : Définit toutes les valeurs de la clé étrangère dans la table référencée à NULL lorsque la ligne correspondante est supprimée de la table parente.
  - **RESTRICT** : Empêche la suppression de la ligne parente si des lignes liées existent dans la table référencée.

- ON UPDATE : Cette option détermine ce qui se passe dans la table référencée lorsque la valeur de la clé primaire de la table parente est mise à jour. Les options courantes sont :
  - CASCADE : Met à jour automatiquement toutes les valeurs de la clé étrangère dans la table référencée qui sont liées à la clé primaire mise à jour dans la table parente.
  - SET NULL : Définit toutes les valeurs de la clé étrangère dans la table référencée à NULL lorsque la valeur de la clé primaire correspondante est mise à jour dans la table parente.
  - RESTRICT : Empêche la mise à jour de la clé primaire si des lignes liées existent dans la table référencée.

En définissant ces options, vous pouvez spécifier comment les modifications apportées aux données dans une table doivent être gérées lorsque des données liées existent dans une autre table. Cela permet d'assurer l'intégrité référentielle de la base de données et de garantir que les données restent cohérentes.

On va pouvoir maintenant créer des relations entre nos clés primaires et nos clés étrangères directement dans l'onglet "concepteur" de phpMyAdmin.

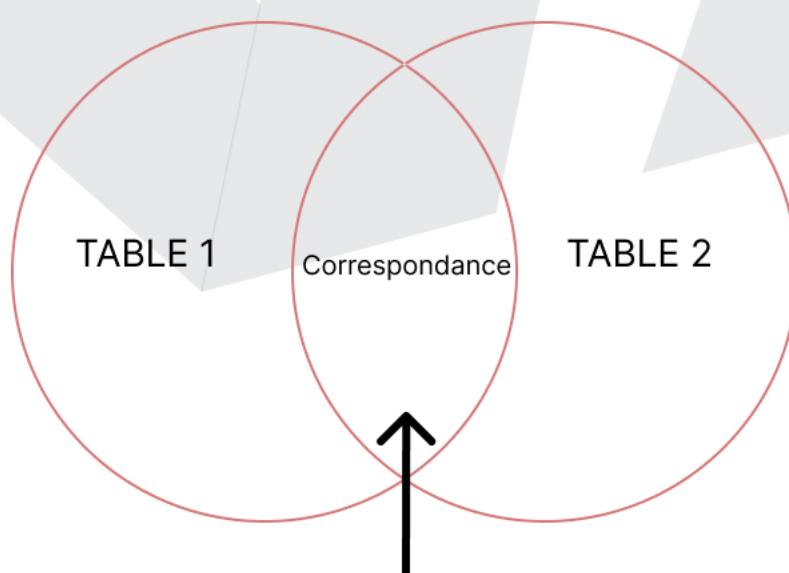


En SQL, les jointures internes (inner joins) sont utilisées pour combiner les données de deux ou plusieurs tables en utilisant une colonne commune. Les jointures internes ne renvoient que les lignes qui ont des correspondances dans les deux tables jointes.

La syntaxe de base pour une jointure interne est la suivante :

```
SELECT [colonnes]
FROM [table1]
JOIN [table2] ON [table1].[colonne] =
[table2].[colonne]
```

Dans cette syntaxe, la clause SELECT spécifie les colonnes que vous souhaitez inclure dans le résultat de la jointure. La clause FROM spécifie les tables que vous souhaitez joindre. La clause JOIN spécifie la table que vous souhaitez joindre et la condition ON spécifie les colonnes sur lesquelles les tables seront jointes.



ce que vont va chercher  
avec la jointure interne



Par exemple, pour sélectionner une commande spécifique à partir des correspondances:

```

SELECT c.com_id, cl.client_nom,
cl.client_prenom FROM commandes c JOIN
client cl ON c.client_id = cl.client_id
WHERE c.client_id = 2;
  
```

Un exemple plus complexe pour afficher toutes les informations d'une commande:

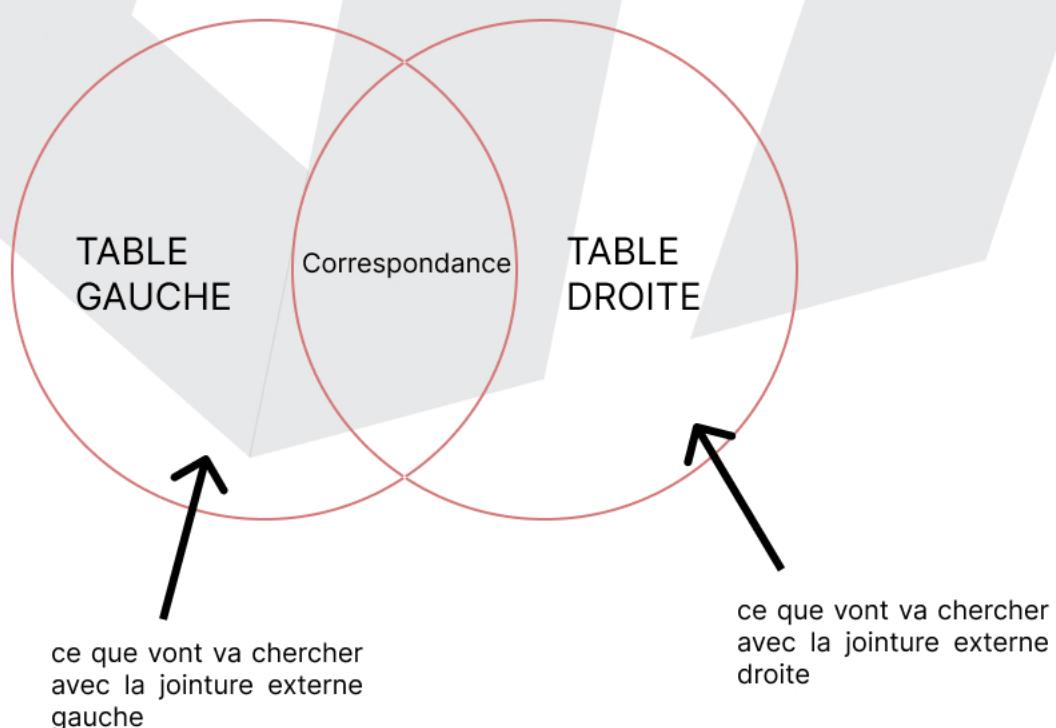
```

SELECT c.com_id, c.com_tarif, c.com_date,
cl.client_id, cl.client_nom, a.art_id,
a.art_nom
FROM commandes c
JOIN client cl ON c.client_id =
cl.client_id
JOIN commande_article ca ON c.com_id =
ca.com_id
JOIN article a ON ca.art_id = a.art_id
WHERE c.client_id = 1;
  
```

Si j'ai une commande sans aucun client\_id alors il me retournera 0

Les jointures externes en SQL permettent de combiner les enregistrements de deux ou plusieurs tables même si certains enregistrements n'ont pas de correspondance dans les autres tables. Il existe deux types de jointures externes : la jointure externe gauche et la jointure externe droite. Voici la syntaxe:

```
SELECT [colonnes]
FROM [table1]
LEFT | RIGHT JOIN [table2] ON [table1].
[colonne] = [table2].[colonne]
```



- Jointure externe gauche :

```
SELECT c.com_id, c.com_tarif,
c.com_date, a.art_nom FROM commandes c LEFT
JOIN client cl ON c.client_id =
cl.client_id LEFT JOIN commande_article ca
ON c.com_id = ca.com_id LEFT JOIN article
a ON ca.art_id = a.art_id;
```

Dans cette requête, nous utilisons la jointure externe gauche pour lier les tables "commande", "client", "commande\_article" et "article".

La table "commande" est la table principale et les autres tables sont liées à elle. La clause LEFT JOIN récupère tous les enregistrements de la table de gauche (ici la table "commande") et les joint avec les enregistrements correspondants de la table de droite (ici la table "client").

Si aucun enregistrement correspondant n'est trouvé dans la table de droite, les valeurs NULL sont retournées pour les colonnes de la table de droite.

- Jointure externe droite :

```
SELECT c.com_id, c.com_tarif, c.com_date,
cl.nom, cl.mail, a.art_nom
FROM commande c
RIGHT JOIN client cl ON c.client_id =
cl.client_id
RIGHT JOIN commande_article ca ON c.com_id =
ca.com_id
RIGHT JOIN article a ON ca.art_id =
a.art_id;
```

Dans cette requête, nous utilisons la jointure externe droite pour lier les tables "commande", "client", "commande\_article" et "article". La table "client" est la table principale et les autres tables sont liées à elle. La clause RIGHT JOIN récupère tous les enregistrements de la table de droite (ici la table "client") et les joint avec les enregistrements correspondants de la table de gauche (ici la table "commande").

Si aucun enregistrement correspondant n'est trouvé dans la table de gauche, les valeurs NULL sont renournées pour les colonnes de la table de gauche.

Notez que ces deux requêtes peuvent retourner des enregistrements en double si les enregistrements ont plusieurs correspondances dans une table. Pour éviter cela, vous pouvez utiliser la clause DISTINCT pour ne récupérer que les enregistrements uniques.

Nous pouvons également récupérer la table de gauche et de droite avec la jonction interne FULL OUTER JOIN

```
SELECT * from TABLE1 full outer join  
TABLE2 on TABLE1.ID + TABLE2.ID
```

L'opérateur UNION en SQL est utilisé pour combiner les résultats de deux ou plusieurs requêtes SELECT en un seul résultat, en éliminant les doublons. Les requêtes utilisées avec l'opérateur UNION doivent avoir le même nombre de colonnes et les colonnes doivent être de types de données compatibles.

La syntaxe générale de l'opérateur UNION est la suivante:

```
SELECT colonne_1, colonne_2, ...,
colonne_n FROM table_1
UNION
SELECT colonne_1, colonne_2, ...,
colonne_n FROM table_2
```

Exemple:

```
SELECT com_id FROM commandes UNION SELECT  
client_nom, client_prenom FROM client
```

Ceci marquera une erreur. Pour remédier à cela on fera:

```
SELECT com_id, com_tarif FROM commandes  
UNION SELECT client_nom, client_prenom  
FROM client
```

Il faut le même nombre de colonnes par table. J'obtiendrais toutes les commandes et les clients qui n'ont pas passé de commande également. Si on veut afficher également les doublons nous rajoutons ALL:

```
SELECT com_id, com_tarif FROM commandes  
UNION ALL SELECT client_nom, client_prenom  
FROM client
```

En SQL, la fonction COALESCE() permet de renvoyer la première valeur non nulle parmi une liste de valeurs données en entrée.

```
SELECT c.com_id, cl.client_nom,  
cl.client_prenom FROM commandes c LEFT  
OUTER JOIN client cl ON c.client_id =  
cl.client_id UNION SELECT c.com_id,  
cl.client_nom, cl.client_prenom FROM  
commandes c RIGHT OUTER JOIN client cl ON  
c.client_id = cl.client_id;
```

La fonction prend en entrée une liste de valeurs séparées par des virgules, et renvoie la première valeur non nulle de cette liste. Si toutes les valeurs sont nulles, la fonction renvoie NULL.

Voici un exemple d'utilisation de la fonction COALESCE() :

```
SELECT COALESCE(c.com_id, 'Pas de  
commande') AS cont_id ,  
COALESCE(cl.client_id, 'Pas de commande')  
AS nom_id  
FROM commandes c  
LEFT OUTER JOIN client cl  
ON c.client_id = cl.client_id  
UNION  
SELECT COALESCE(c.com_id, 'Pas de  
commande') AS cont_id ,  
COALESCE(cl.client_id, 'Pas de commande')  
AS nom_id  
FROM commandes c  
RIGHT OUTER JOIN client cl  
ON c.client_id = cl.client_id;
```

Maintenant réalisons avec tout sa une grande requêtes . Nous allons récupère pour chaque commande le nom et le prénom du client et l'article commandé pour cette commande pour le client\_id numéro 1. Rajoutons dabord art\_id à la table commande et com\_id à la table article ensuite:

```
SELECT c.com_id, c.com_tarif, c.com_date,
cl.client_nom, cl.client_prenom, a.art_nom
FROM commandes c
INNER JOIN client cl ON c.client_id =
cl.client_id
INNER JOIN commande_article ca
ON c.com_id = ca.com_id
INNER JOIN article a
ON c.art_id = a.art_id
WHERE c.com_id = 2;
```

Une jointure naturelle en SQL est une opération qui relie deux tables en utilisant leurs colonnes ayant le même nom. Cela signifie que les colonnes qui ont le même nom et le même type de données sont automatiquement jointes lorsqu'une requête est exécutée.

Lorsque vous utilisez une jointure naturelle en SQL, le résultat renvoyé est une combinaison des enregistrements des deux tables qui correspondent sur la colonne qui a le même nom dans les deux tables.

Voici la syntaxe:

```
SELECT * FROM table1 NATURAL JOIN table2;
```

Par exemple:

```
SELECT *
FROM commandes
NATURAL JOIN client
```

Cela fonctionne comme inner join il va afficher les correspondances par rapport aux clés primaires et au clés étrangères.

+ Options

	<input type="button" value="←"/> <input type="button" value="→"/>		▼	catID	nom	parID
<input type="checkbox"/>		Éditer		Copier		Supprimer
<input type="checkbox"/>		Éditer		Copier		Supprimer
<input type="checkbox"/>		Éditer		Copier		Supprimer
<input type="checkbox"/>		Éditer		Copier		Supprimer

Les jointures automatiques (ou implicites) en SQL sont des jointures qui sont déterminées automatiquement par le moteur de base de données en fonction des relations entre les tables.

Lorsqu'une requête SQL inclut deux ou plusieurs tables, le moteur de base de données peut automatiquement déterminer la façon de lier ces tables en utilisant les colonnes de clé commune entre elles.

Voici la syntaxe:

```
SELECT * FROM table1 [ ] JOIN table2 ON
1.id = 2.id;
```

Par exemple:

```
SELECT a.nom AS catParent, b.nom AS
catEnfant FROM categories a INNER JOIN
categories b ON a.catID = b.parID;
```

Les jointures automatiques sont souvent utilisées pour simplifier les requêtes SQL, car elles évitent la nécessité de spécifier explicitement la clause JOIN avec les colonnes correspondantes.

La commande CROSS JOIN en SQL permet de combiner chaque ligne d'une table avec chaque ligne d'une autre table. Elle crée un produit cartésien entre les deux tables, c'est-à-dire toutes les combinaisons possibles de chaque ligne de la première table avec chaque ligne de la seconde table.

La syntaxe de la commande CROSS JOIN est la suivante :

```
SELECT *
FROM table1
CROSS JOIN table2;
```

Par exemple:

+ Options		coul_id	coul
<input type="checkbox"/>	Éditer	 Copier	 Supprimer
<input type="checkbox"/>	Éditer	 Copier	 Supprimer
<input type="checkbox"/>	Éditer	 Copier	 Supprimer

+ Options		car_id	car_nom
<input type="checkbox"/>	Éditer	 Copier	 Supprimer
<input type="checkbox"/>	Éditer	 Copier	 Supprimer

Je voudrais afficher toutes les couleurs possible pour chaque voiture:

```
SELECT v.car_nom, c.coul FROM car v CROSS
JOIN couleurs c
```

Il existe 3 manières d'écrire des commentaire en SQL:

Méthode 1 avec les --:

```
SELECT v.car_nom, c.coul -- je suis un
commentaire
FROM car v CROSS JOIN couleurs c
```

Méthode 2 avec le #:

```
SELECT v.car_nom, c.coul # je suis un
commentaire
FROM car v CROSS JOIN couleurs c
```

Méthode 3 avec /\* \*/ pour les commentaire à plusieurs lignes:

```
SELECT v.car_nom, c.coul /* je suis un
commentaire */
FROM car v CROSS JOIN couleurs c
```



# laminutedecode.io

@LAMINUTEDECODE

