Les **quatre types de jointures** en SQL sont :

**INNER JOIN** – Récupère **uniquement** les correspondances dans les deux tables.  
**LEFT JOIN (LEFT OUTER JOIN)** – Récupère **toutes les lignes de la table de gauche**, même si elles n'ont pas de correspondance dans la table de droite.  
**RIGHT JOIN (RIGHT OUTER JOIN)** – Récupère **toutes les lignes de la table de droite**, même si elles n'ont pas de correspondance dans la table de gauche.  
**FULL JOIN (FULL OUTER JOIN)** – Récupère **toutes les lignes des deux tables**, en mettant NULL si aucune correspondance.

**Exemple de Base**

On va utiliser ces deux tables :

**Table utilisateurs**

| **id** | **prénom** | **nom** |
| --- | --- | --- |
| 1 | Alice | Dupont |
| 2 | Bob | Martin |
| 3 | Charlie | Durand |
| 4 | David | Moreau |

**Table photos**

| **id** | **url** | **photographe** |
| --- | --- | --- |
| 1 | photo1.jpg | 1 |
| 2 | photo2.jpg | 3 |
| 3 | photo3.jpg | 1 |
| 4 | photo4.jpg | 5 (existe pas) |

**INNER JOIN**

**Retourne uniquement les utilisateurs qui ont pris des photos.**  
Si un utilisateur **n'a jamais pris de photo, il ne sera pas affiché**.

SELECT u.prenom, u.nom, p.url

FROM utilisateurs u

INNER JOIN photos p ON u.id = p.photographe;

**Résultat :**

| **prénom** | **nom** | **url** |
| --- | --- | --- |
| Alice | Dupont | photo1.jpg |
| Alice | Dupont | photo3.jpg |
| Charlie | Durand | photo2.jpg |

**David n'apparaît pas car il n'a pris aucune photo.**  
 **photo4.jpg ne s'affiche pas car son photographe = 5, et l'utilisateur 5 n'existe pas.**

**LEFT JOIN (LEFT OUTER JOIN)**

**Garde tous les utilisateurs, même ceux qui n'ont pas pris de photo.**  
Si un utilisateur **n'a pas de photo, on met NULL**.

SELECT u.prenom, u.nom, p.url

FROM utilisateurs u

LEFT JOIN photos p ON u.id = p.photographe;

**Résultat :**

| **prénom** | **nom** | **url** |
| --- | --- | --- |
| Alice | Dupont | photo1.jpg |
| Alice | Dupont | photo3.jpg |
| Bob | Martin | NULL |
| Charlie | Durand | photo2.jpg |
| David | Moreau | NULL |

**Bob et David sont affichés, même s'ils n'ont pas pris de photo.**  
 **photo4.jpg ne s'affiche pas car son photographe (id=5) n'existe pas dans utilisateurs.**

**RIGHT JOIN (RIGHT OUTER JOIN)**

**Garde toutes les photos, même si elles n'ont pas de photographe existant.**  
Si une photo **n'a pas de photographe valide, on met NULL**.

SELECT u.prenom, u.nom, p.url

FROM utilisateurs u

RIGHT JOIN photos p ON u.id = p.photographe;

**Résultat :**

| **prénom** | **nom** | **url** |
| --- | --- | --- |
| Alice | Dupont | photo1.jpg |
| Alice | Dupont | photo3.jpg |
| Charlie | Durand | photo2.jpg |
| NULL | NULL | photo4.jpg |

**photo4.jpg est affichée même si le photographe n'existe pas.**  
**Bob et David disparaissent car ils n'ont pas pris de photo.**

**FULL JOIN (FULL OUTER JOIN)**

**Garde tout : tous les utilisateurs ET toutes les photos.**  
Si une ligne **n'a pas de correspondance, elle aura NULL**.

SELECT u.prenom, u.nom, p.url

FROM utilisateurs u

FULL JOIN photos p ON u.id = p.photographe;

**Résultat :**

| **prénom** | **nom** | **url** |
| --- | --- | --- |
| Alice | Dupont | photo1.jpg |
| Alice | Dupont | photo3.jpg |
| Bob | Martin | NULL |
| Charlie | Durand | photo2.jpg |
| David | Moreau | NULL |
| NULL | NULL | photo4.jpg |

**Tout le monde est affiché.**  
**photo4.jpg est affichée même si le photographe n'existe pas.**  
**Bob et David sont affichés même s'ils n'ont pas pris de photo.**

**Récapitulatif des Différences**

| **Jointure** | **Garde tous les utilisateurs ?** | **Garde toutes les photos ?** |
| --- | --- | --- |
| **INNER JOIN** | ❌ Non (seulement ceux avec des photos) | ❌ Non (seulement celles avec un photographe existant) |
| **LEFT JOIN** | ✅ Oui | ❌ Non (seulement celles avec un photographe existant) |
| **RIGHT JOIN** | ❌ Non (seulement ceux qui ont pris des photos) | ✅ Oui |
| **FULL JOIN** | ✅ Oui | ✅ Oui |

**Quand Utiliser Quelle Jointure ?**

* **INNER JOIN** : Si tu veux **seulement les correspondances exactes**.  
  → Exemple : Voir quels utilisateurs ont liké une photo.
* **LEFT JOIN** : Si tu veux **tous les utilisateurs même ceux sans correspondance**.  
  → Exemple : Trouver les utilisateurs qui n'ont jamais liké de photo.
* **RIGHT JOIN** : Si tu veux **toutes les photos, même celles sans utilisateur**.  
  → Exemple : Lister toutes les photos même si le photographe a été supprimé.
* **FULL JOIN** : Si tu veux **tout afficher, avec des NULL pour les données manquantes**.  
  → Exemple : Faire une analyse complète de tous les utilisateurs et toutes les photos.

**COUNT**

**Quand l'utiliser précisément :** COUNT est une fonction d'agrégation qui compte le nombre de lignes ou de valeurs non NULL dans une colonne. Utilisez COUNT lorsque vous avez besoin de :

* Déterminer le volume total d'enregistrements
* Compter des occurrences spécifiques qui répondent à certains critères
* Analyser la distribution des données à travers votre base de données

**Exemple 1 :** Compter le nombre total de photos dans la base de données

SELECT COUNT(\*) AS nombre\_total\_photos FROM photos;

Ce code retourne un seul chiffre représentant le nombre total d'enregistrements dans la table photos (14 dans votre cas).

**Exemple 2 :** Compter le nombre de photos ayant une catégorie assignée versus celles sans catégorie

SELECT

SUM(CASE WHEN categorie\_id IS NOT NULL THEN 1 ELSE 0 END) AS photos\_avec\_categorie,

SUM(CASE WHEN categorie\_id IS NULL THEN 1 ELSE 0 END) AS photos\_sans\_categorie

FROM photos;

Ce code utilise COUNT implicitement dans des expressions conditionnelles pour segmenter les photos.

**Exemple 3 :** Compter le nombre de likes reçus pour chaque photo

SELECT p.id AS photo\_id, p.url, COUNT(l.id) AS nombre\_likes

FROM photos p

LEFT JOIN likes l ON p.id = l.photo\_id

GROUP BY p.id, p.url

ORDER BY nombre\_likes DESC;

Ce code compte les likes pour chaque photo, y compris celles qui n'ont reçu aucun like (grâce au LEFT JOIN).

**GROUP BY**

**Quand l'utiliser précisément :** GROUP BY est utilisé pour segmenter vos données en groupes selon les valeurs d'une ou plusieurs colonnes. Utilisez GROUP BY quand vous devez :

* Analyser des données agrégées par catégories ou dimensions
* Créer des statistiques récapitulatives sur des sous-ensembles de données
* Préparer des données pour des analyses comparatives entre différents groupes

**Exemple 1 :** Nombre de photos publiées par chaque photographe, avec leurs noms

SELECT u.id, u.prenom, u.nom, COUNT(p.id) AS nombre\_photos

FROM utilisateurs u

LEFT JOIN photos p ON u.id = p.photographe

GROUP BY u.id, u.prenom, u.nom

ORDER BY nombre\_photos DESC;

Ce code regroupe les photos par photographe et affiche combien chacun en a publié.

**Exemple 2 :** Distribution des photos par catégorie, montrant le pourcentage du total

SELECT

c.nom AS categorie,

COUNT(p.id) AS nombre\_photos,

ROUND(COUNT(p.id) \* 100.0 / (SELECT COUNT(\*) FROM photos), 2) AS pourcentage

FROM categories c

LEFT JOIN photos p ON c.id = p.categorie\_id

GROUP BY c.id, c.nom

ORDER BY nombre\_photos DESC;

Ce code montre comment les photos sont distribuées entre les différentes catégories.

**Exemple 3 :** Nombre moyen de likes reçus par catégorie de photo

SELECT

c.nom AS categorie,

COUNT(DISTINCT p.id) AS nombre\_photos,

COUNT(l.id) AS total\_likes,

ROUND(COUNT(l.id) \* 1.0 / COUNT(DISTINCT p.id), 2) AS moyenne\_likes\_par\_photo

FROM categories c

JOIN photos p ON c.id = p.categorie\_id

LEFT JOIN likes l ON p.id = l.photo\_id

GROUP BY c.id, c.nom

ORDER BY moyenne\_likes\_par\_photo DESC;

Ce code calcule plusieurs métriques en regroupant par catégorie de photo.

**HAVING**

**Quand l'utiliser précisément :** HAVING filtre les résultats après que les groupes ont été formés et les agrégations calculées. Utilisez HAVING lorsque vous devez :

* Filtrer des résultats basés sur des calculs d'agrégation
* Appliquer des conditions sur des groupes entiers plutôt que sur des lignes individuelles
* Réduire les résultats d'agrégation selon des seuils ou critères spécifiques

**Exemple 1 :** Trouver les photographes productifs (ayant publié plus de 2 photos)

SELECT

u.prenom,

u.nom,

COUNT(p.id) AS nombre\_photos

FROM utilisateurs u

JOIN photos p ON u.id = p.photographe

GROUP BY u.id, u.prenom, u.nom

HAVING COUNT(p.id) > 2

ORDER BY nombre\_photos DESC;

Ce code identifie les photographes les plus actifs sur la plateforme.

**Exemple 2 :** Trouver les catégories populaires qui reçoivent un ratio élevé de likes par photo

SELECT

c.nom AS categorie,

COUNT(DISTINCT p.id) AS nombre\_photos,

COUNT(l.id) AS nombre\_likes,

ROUND(COUNT(l.id) \* 1.0 / COUNT(DISTINCT p.id), 2) AS ratio\_likes\_par\_photo

FROM categories c

JOIN photos p ON c.id = p.categorie\_id

JOIN likes l ON p.id = l.photo\_id

GROUP BY c.id, c.nom

HAVING COUNT(l.id) / COUNT(DISTINCT p.id) > 1

ORDER BY ratio\_likes\_par\_photo DESC;

Ce code identifie les catégories qui reçoivent en moyenne plus d'un like par photo.

**Exemple 3 :** Trouver les utilisateurs qui aiment des photos de plusieurs photographes différents

SELECT

u.prenom,

u.nom,

COUNT(DISTINCT p.photographe) AS nombre\_photographes\_differents

FROM utilisateurs u

JOIN likes l ON u.id = l.utilisateur\_id

JOIN photos p ON l.photo\_id = p.id

GROUP BY u.id, u.prenom, u.nom

HAVING COUNT(DISTINCT p.photographe) > 1

ORDER BY nombre\_photographes\_differents DESC;

Ce code identifie les utilisateurs aux goûts diversifiés

• Un troisième exemple : -- Trouver les employés qui gagnent au moins la moyenne salariale -- de leur département et qui sont en minorité de genre dans leur département.

SELECT nom, prenom

FROM employe AS emp

WHERE salaire >= (

SELECT AVG(es.salaire)

FROM employe AS es

WHERE es.departement = emp.departement) AND genre = (

SELECT eg.genre FROM employe AS eg WHERE eg.departement = emp.departement GROUP BY eg.genre ORDER BY COUNT(\*) ASC LIMIT 1);