SQL POUR LES NULS

TEAM BLAUGRANAS

SOMMAIRE

SQL, C'EST QUOI ?

(STRUCTURED QUERY LANGUAGE)

c'est un langage de programmation

- gérer
- manipuler
- stocker

des données relationnelles

ET DANS NOTRE FUTUR?

1. Analyse
2. Exploration
3. Extraction
4. Informations pertinentes

RAPPELS DE BASE

Les types de données

INTEGER

VARCHAR

DATE

FLOAT

BOOLEAN

BLOB

TIMESTAMP

SERIAL

OPÉRATIONS CRUD

CREATE

READ

UPDATE

DELETE

- Le signe égal (=) sert à choisir des choses qui sont exactement pareilles.

```
SELECT * FROM Livre WHERE titre = 'Bienvenue;
```

- Le signe différent (<> ou !=) sert à choisir des choses qui ne sont pas exactement les mêmes.

```
SELECT * FROM Livre WHERE titre <> 'Bienvenue';
```

- Le signe "supérieur à" (>) sert à choisir des choses qui sont plus grandes

```
SELECT * FROM Auteur WHERE date_naissance > '2000-01-01';
```

- > pour des choses plus petites

```
SELECT * FROM Auteur WHERE date_naissance < '2000-01-01';
```

- Supérieur ou égal à >= / inférieur ou égal <=

- Plage (BETWEEN ... AND ...) : Lorsqu'on utilise cela permet de choisir des choses qui sont dans une certaine étendue ou une période

```
SELECT * FROM Auteur WHERE date_naissance BETWEEN '1950-01-01' AND '1970-12-31';
```

- Recherche basée sur un motif (LIKE) :

```
SELECT * FROM Livre WHERE titre LIKE '%Shakespeare%';
```

- IN : Pour indiquer les Valeurs possibles dans une colonne

```
SELECT * FROM Livre WHERE genre_id IN (4, 5, 6);
```

- IS NULL et IS NOT NULL

Vérification des valeurs Null ou non null

```
SELECT * FROM Livre WHERE date_publication IS NULL;
```

- AND et OR :

Assembler des conditions

- NOT :

Négation de condition

```
SELECT * FROM Livre WHERE NOT genre_id = 1;
```

GROUPER ET AGGRÉGER

ORDER BY : Permet de trier les résultats.
`SELECT * FROM Livre ORDER BY date_publication DESC;`
affiche tous les livres triés par date de publication de manière décroissante.

GROUP BY : Regroupe les résultats en fonction d'une colonne.
`SELECT genre_id, COUNT(*) FROM Livre GROUP BY genre_id;`
compte le nombre de livres par genre.

HAVING : Filtre les résultats après l'utilisation de GROUP BY.
`SELECT genre_id, COUNT(*) FROM Livre GROUP BY genre_id HAVING COUNT(*) > 5;`
affiche les genres avec plus de 5 livres.

FONCTIONS D'AGGRÉGATION

Ces fonctions sont utilisé avec la clause GROUP BY

COUNT : Compte le nombre d'enregistrements dans un groupe.
`SELECT COUNT(*) FROM Livre;` donne le nombre total de livres

SUM : Calcule la somme d'une colonne numérique.
`SELECT SUM(prix) FROM Commande;` donne la somme totale des prix des commandes.

AVG : Calcule la moyenne d'une colonne numérique.
`SELECT AVG(note) FROM Avis;` donne la note moyenne des avis.

MIN : Retourne la valeur minimale d'une colonne.
`SELECT MIN(date_publication) FROM Livre;` donne la date de publication du livre le plus ancien.

MAX : Retourne la valeur maximale d'une colonne.
`SELECT MAX(date_publication) FROM Livre;` donne la date de publication du livre le plus

REQUÊTES INTERMÉDIAIRES

DIFFERENTS OPERATIONS D'ENSEMBLE

- **UNION**

Il permet de fusionne les résultats de deux requêtes en un seul ensemble de lignes

Ex:

```
SELECT id_client, nom, email
       FROM clients1
           UNION
SELECT id_client, nom, email
       FROM clients2;
```

- **UNION ALL**

L'UNION ALL fait le meme chose que UNION, mais en conservant les doublons.

Ex:

```
SELECT id_client, nom, email
       FROM clients1
         UNION ALL
SELECT id_client, nom, email
       FROM clients2;
```

DIFFERENTS OPERATIONS D'ENSEMBLE

- **EXCEPT**

Il retourne toutes les lignes de la première requête qui ne figurent pas dans la seconde requête.

Ex:

```
SELECT id_client, nom, email FROM clients1
                EXCEPT
SELECT id_client, nom, email FROM clients2;
```

- **INTERSECT**

Il retourne toutes les lignes communes aux deux requêtes.

Ex:

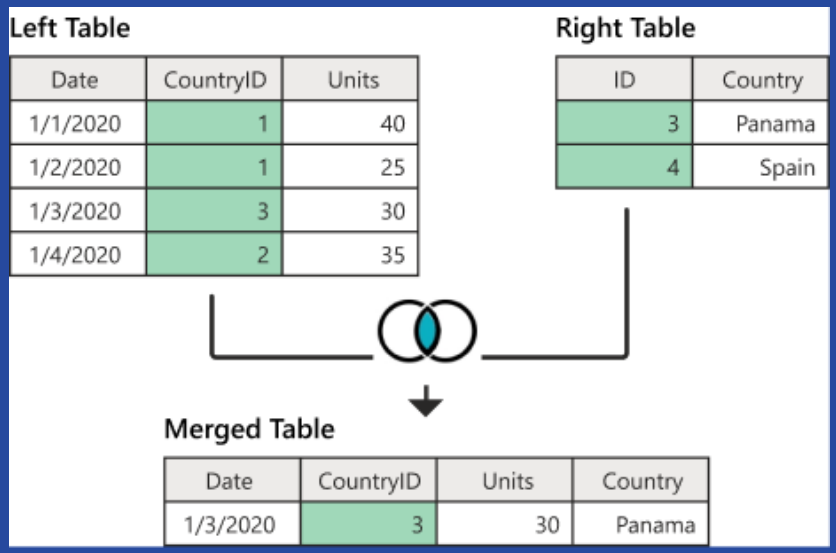
```
SELECT id_client, nom, email FROM clients1
               INTERSECT
SELECT id_client, nom, email FROM clients2;
```

LES JOINTURES

Les jointures en SQL permettent de combiner des données de deux ou plusieurs tables en fonction d'une condition commune.

- **INNER JOIN**

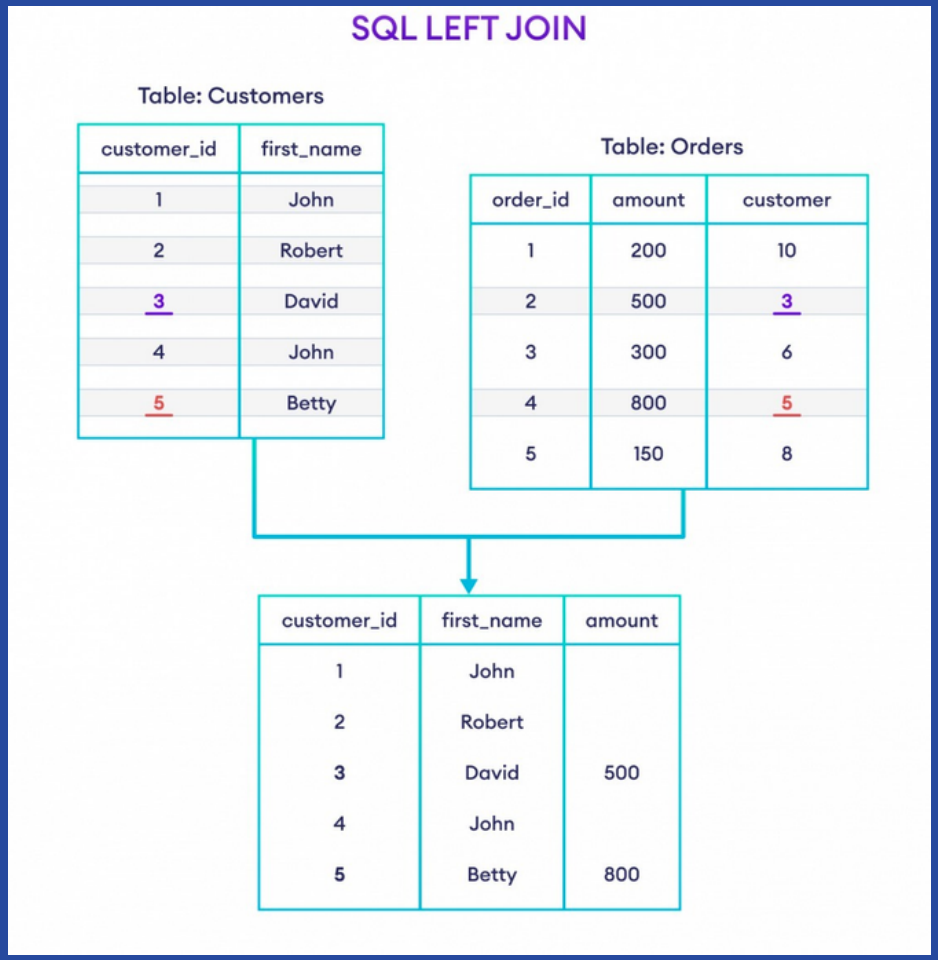
Retourne les lignes pour lesquelles il existe une correspondance dans les deux tables.




```
SELECT employes.nom, employes.salaire,
departements.nom_departement
FROM employes
INNER JOIN departements
ON employes.departement_id = departements.departement_id;
```

- # LEFT jOIN

retourne toutes les lignes de la table de gauche et les lignes correspondantes de la table de droite.

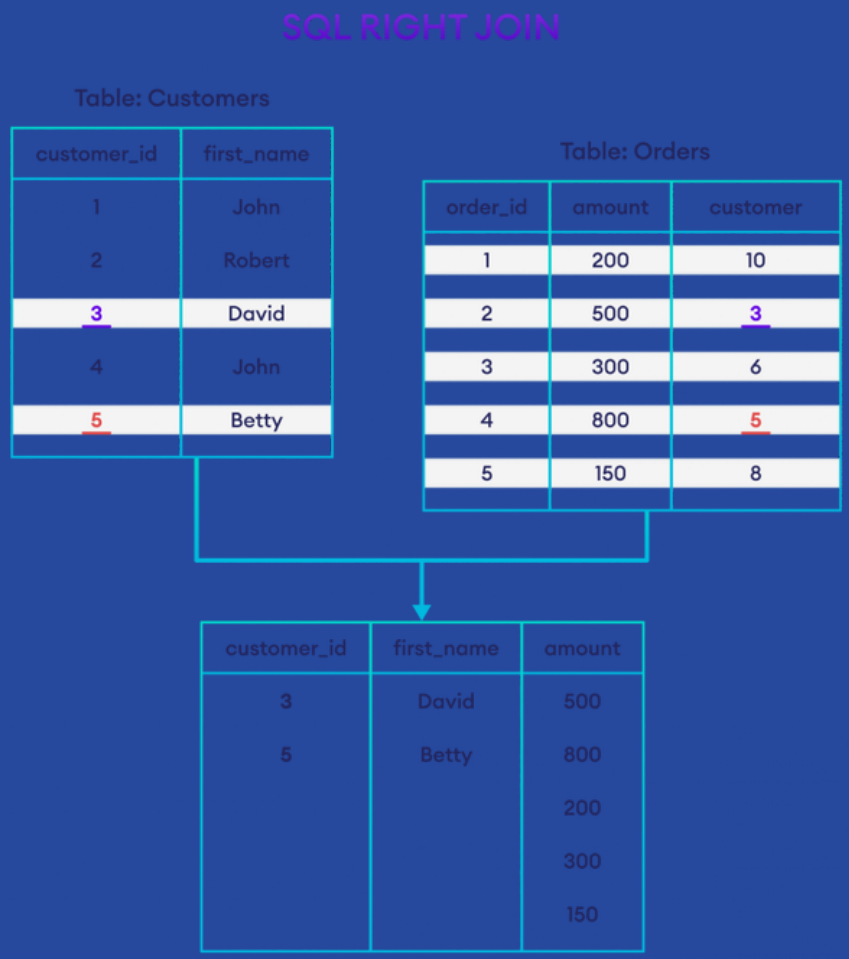


Ex

```
SELECT *
FROM Clients
LEFT JOIN Commandes
ON Clients.ClientID = Commandes.ClientID;
```

RIGHT JOIN

Cette jointure retourne toutes les lignes de la table de droite et les lignescorrespondantes de la table de gauche.




```
SELECT employes.nom, employes.salaire,
departements.nom_departement
FROM employes
RIGHT JOIN departements ON
employes.departement_id =
departements.departement_id;
```

FULL JOIN

FULL JOIN retourne les lignes lorsque la condition est vraie dans l'une des tables.

```
SELECT employes.nom, employes.salaire,
departements.nom_departement
FROM employes
```

Name	# of Friends
Matt	300
Lisa	500
Jeff	600
Sarah	400

Name	# of connections
Matt	500
Lisa	200
Sarah	100
Louis	300

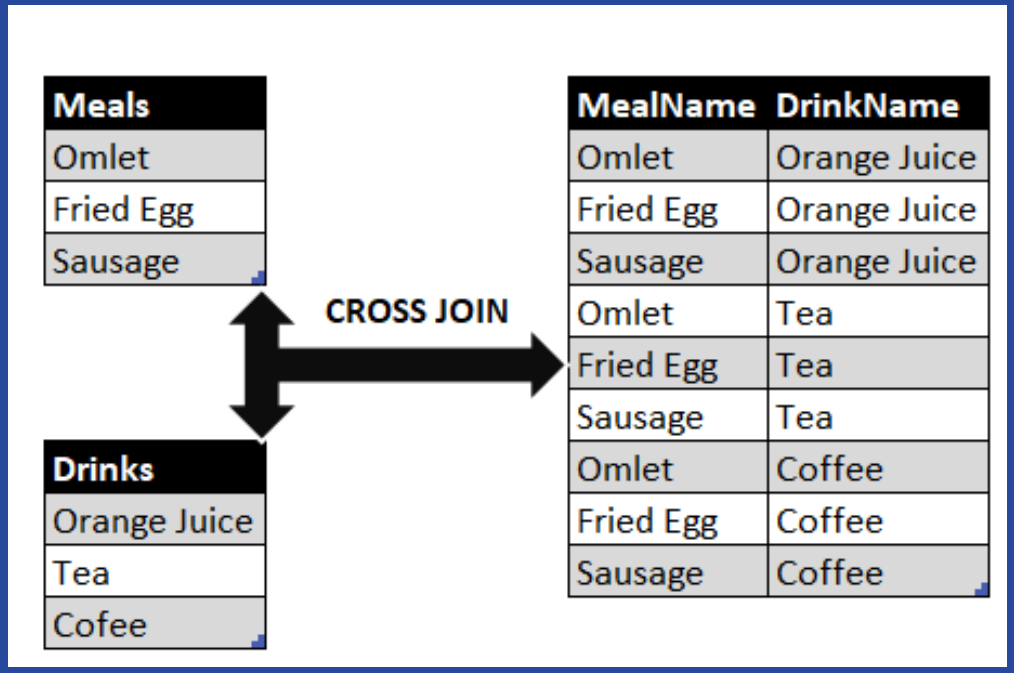
8

facebook and linkedin JOINed

facebook.Name	facebook.# of Friends	linkedin.Name	linkedin.# of connections
Matt	300	Matt	500
Lisa	500	Lisa	200
Jeff	600	*Null*	*Null*
Sarah	400	Sarah	100
Null	*Null*	Louis	300

CROSS JOIN

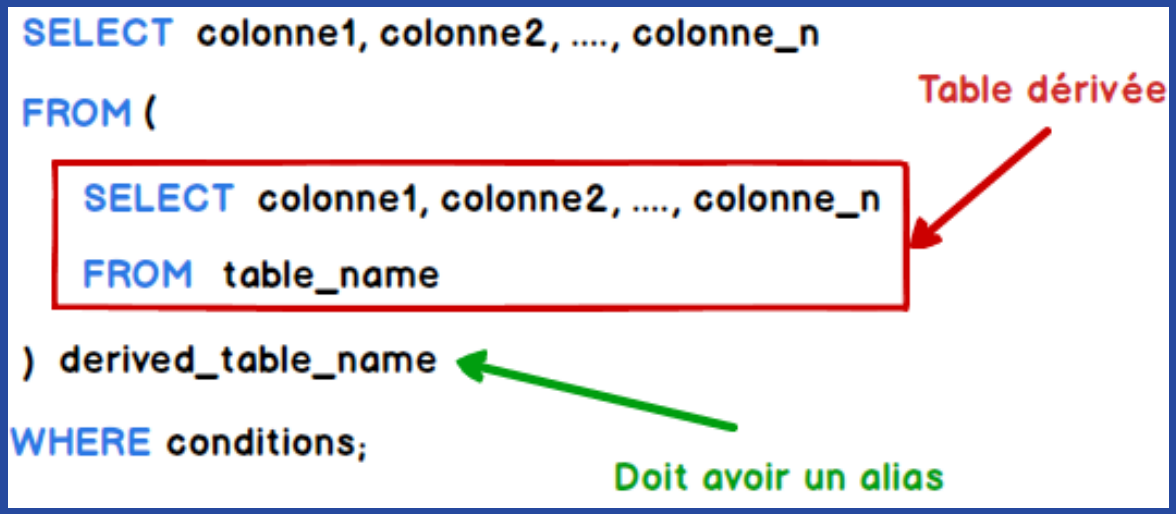
Il n'y a pas de condition de correspondance explicite entre les tables comme pour les autres types de jointures. Par conséquent, si une table a n lignes et l'autre table a m lignes, le résultat de la jointure croisée aura n x m lignes.




```
SELECT employes.nom, employes.salaire,
departements.nom_departement
FROM employes
CROSS JOIN departements;
```

Sous-requetes

Une sous-requête est une requête imbriquée à l'intérieur d'une autre requête. Elle permet d'obtenir des données qui seront utilisées dans la requête principale comme une condition de filtrage.




```
SELECT nom
FROM employes
WHERE departement_id = (
    SELECT departement_id
    FROM departements
    WHERE nom_departement =
    'Informatique'
);
```

Les Vues

Une vue est une requête SQL sauvegardée en tant qu'objet de base de données,permettant ainsi de traiter cette requête comme s'il s'agissait d'une table à part entière.
Pour créer une vue, on utilise généralement la syntaxe suivante

Création

```
CREATE VIEW nom_de_la_vue AS
        SELECT ...
         FROM ...
        WHERE ...;
   -- Utilisation de la vue
SELECT * FROM vue_informatique;
```

Exemple:

```
CREATE VIEW vue_informatique AS
      SELECT nom, salaire
        FROM employes
   WHERE departement_id = (
       SELECT departement_id
         FROM departements
WHERE nom_departement = 'Informatique'
                );
   -- Utilisation de la vue
SELECT * FROM vue_informatique;
```

Case

CASE est une instruction conditionnelle en SQL qui permet de retourner une valeur sur une colonne en fonction d'une ou plusieurs conditions. C'est équivalent à l'utilisation d’ if-else dans d'autres langages de programmation.

Exemple:

```
SELECT
id_commande,  montant, date_commande,statut,
CASE
WHEN montant > 200.00 AND statut = 'En cours' THEN 'Commande importante en cours'
WHEN montant > 200.00 AND statut = 'Livré' THEN 'Commande importante livrée'
WHEN montant <= 200.00 AND statut = 'En cours' THEN 'Commande régulière en cours'
WHEN montant <= 200.00 AND statut = 'Livré' THEN 'Commande régulière livrée'
ELSE 'Statut non géré'
END AS categorie_commande
FROM commandes;
```

MANIPULATION DE TEXTE ET DE DATES

TEXTE

|| ' ' || : La concaténation est l'action de joindre deux chaînes de caractères en une seule.

```
SELECT prenom_emp || ' ' || nom_emp AS nom_complet
FROM employe;
```

POSITION : Pour trouver la position d'un texte spécifique dans une chaîne.

```
SELECT POSITION('ohn' IN prenom_emp) AS position_prenom
FROM employe;
```

SUBSTRING : Pour extraire une partie spécifique d'une chaîne

```
SELECT SUBSTRING(titre_poste FROM 1 FOR POSITION(': ' IN titre_poste) - 1) AS sous_titre
FROM employe
WHERE titre_poste LIKE '%: %';
```

UPPER / LOWER : Pour normaliser les données textuelles pour la comparaison ou l'affichage.

```
SELECT UPPER(nom_departement) AS departement_majuscule
FROM employe;
SELECT LOWER(nom_departement) AS departement_minuscule
FROM employe;
```

TEXTE

TRIM : Pour supprimer les espaces inutiles

```
SELECT TRIM(nom_emp) AS nom_sans_espaces
FROM employe;
```

LENGTH : Pour obtenir la longueur d'une chaîne de caractères.

```
SELECT titre_poste, LENGTH(titre_poste) AS longueur_titre
FROM employe;
```

REPLACE : Pour remplacer toutes les occurrences d'une sous-chaîne par une autre.

```
SELECT REPLACE(titre_poste, 'ancien', 'nouveau') AS titre_modifie
FROM employe;
```

INITCAP : Pour mettre en majuscule la première lettre de chaque mot dans une chaîne.

```
SELECT INITCAP(titre_poste) AS titre_stylise
FROM employe;
```

DATES

CURRENT_DATE / CURRENT_TIMESTAMP : Ces fonctions fournissent des informations en temps réel sur la date et l'heure actuelles.

```
SELECT CURRENT_DATE;
SELECT CURRENT_TIMESTAMP;
```

EXTRACT : Pour extraire des parties spécifiques d'une date

```
SELECT EXTRACT(YEAR FROM date_embauche) AS annee_embauche
FROM employe;
```

INTERVAL : Pour ajouter ou soustraire des intervalles de temps.

```
SELECT nom_emp, prenom_emp, date_naissance + INTERVAL '1 year' AS
prochain_anniversaire
FROM employe;
```

AGE : Pour calculer l'âge (la différence) entre deux dates.

```
SELECT prenom, nom, AGE(CURRENT_DATE,
date_naissance) AS age_actuel
FROM auteur;
```

D A T E S

DATE_TRUNC : Pour tronquer une date ou une heure à une unité spécifique

```
SELECT titre_poste, DATE_TRUNC('year', date_embauche) AS debut_annee_embauche
FROM employe;
```

DATE_PART : Une autre façon d'extraire une partie d'une date ou d'une heure.

```
SELECT titre_poste, DATE_PART('month', date_embauche) AS mois_embauche
FROM employe;
```

Formatage des dates :

```
SELECT TO_CHAR(date_publication, 'DD/MM/YYYY') AS
date_formattee
FROM livre;
```

Conversion de chaînes en dates

```
SELECT TO_DATE('01/01/2021', 'DD/MM/YYYY') AS date_convertie;
```

Requêtes avancées

Requête corrélée

1. une requête qui sélectionne des employés et une sous-requête qui compte le nombre d'ordinateurs attribués à chaque employé, la sous-requête peut être corrélée à la requête externe en utilisant une colonne commune.

```
SELECT employee_id,
       employee_name,
       (SELECT COUNT(*) FROM computers WHERE employee_id =
employees.employee_id) AS computer_count
FROM employees;
```

Requête imbriquée

Trouver les genres qui ont plus de 5 films

```
SELECT nom_genre
FROM Genre
WHERE genre_id IN (
    SELECT genre_id
    FROM Film
    GROUP BY genre_id
    HAVING COUNT(idFilm) > 5
);
```

Expressions de table communes (CTE)

Exemple : Trouver les réalisateurs qui ont fait plus de 5 films

```
WITH RealisateurPlusDe5Films AS (
    SELECT r.nom, r.prenom, COUNT(f.idFilm) AS nbr_films
    FROM Film f
    JOIN Realisateur r ON f.idRealisateur = r.idRealisateur
    GROUP BY r.nom, r.prenom
    HAVING COUNT(f.idFilm) > 5
)

SELECT *
FROM RealisateurPlusDeFilms;
```

Fonctions de fenêtrage (ou partitionnement)

Classer les films selon leur date de publication et utiliser la fonction ROW_NUMBER()

```
SELECT titre, date_publication,
    ROW_NUMBER() OVER (ORDER BY date_publication) as rang
FROM Film;
```

titre	date_publication	rang
Film 1	2022-01-01	1
Film 2	2022-03-15	2
Film 3	2021-11-30	3
Film 4	2022-05-20	4
Film 5	2022-02-10	5

Fonctions de fenêtrage (ou partitionnement)

```
SELECT r.nom AS nom_realisateur, f.titre, f.date_publication,
    ROW_NUMBER() OVER (PARTITION BY f.idRealisateur ORDER BY
f.date_publication) AS row_num
FROM Film f
JOIN Realisateur r ON f.idRealisateur = r.idRealisateur;
```

nom_realisateur	titre	date_publication	row_num
RealisateurA	FilmA	2022-01-01	1
RealisateurA	FilmC	2021-11-30	2
RealisateurB	FilmB	2022-03-15	1
RealisateurB	FilmD	2022-05-20	2
RealisateurC	FilmE	2022-02-10	1

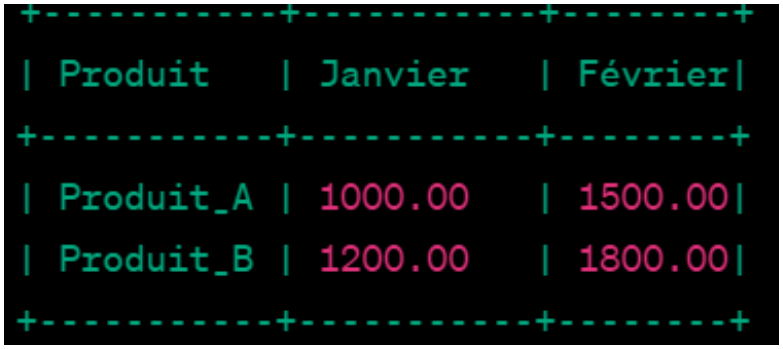
NOUVELLE NOTION: LE PIVOT

Il fait référence à une opération de transformation de données :

- transforme les lignes d'une table en colonnes, créant ainsi une nouvelle structure de tableau croisé dynamique
- l'opération inverse est appelée "unpivot"
- ces opérations sont souvent utilisées pour agréger des données d'une manière plus lisible ou pour effectuer des analyses spécifiques

EXEMPLE D'UTILISATION DU PIVOT

```
SELECT *
FROM (
    SELECT Produit, Mois, Montant
    FROM Ventes
) AS SourceTable
PIVOT (
    SUM(Montant)
    FOR Mois IN ([Janvier],
[Février])
) AS PivotTable;
```



Produit	Janvier	Février
Produit_A	1000.00	1500.00
Produit_B	1200.00	1800.00