DQL

Data Query Language

Introduction aux requêtes DQL

Une requête DQL (Data Query Language) est une requête SQL utilisée pour interroger et récupérer des données depuis une base de données. Elle ne modifie pas les données, mais elle permet de sélectionner, filtrer et trier les données selon les critères spécifiés.

La structure d'une commande DQL (Data Query Language) est la suivante

```
SELECT <liste des données à afficher>
FROM <liste des tables sur lesquelles effectuer des recherches>
WHERE <filtre sur les données à récupérer>
GROUP BY <liste des colonnes à grouper>
HAVING <condition sur les valeurs des fonctions d'agrégation retournées>
ORDER BY <liste des colonnes à ordonner>
LIMIT <nombre de lignes à afficher>
```

L’ordre des clauses est important. Seul la clause `SELECT` est obligatoire.

La clause SELECT

La clause SELECT spécifie les colonnes ou les valeurs que vous souhaitez récupérer dans le résultat de la requête.

La clause suivante retourne le résultat d'un calcul

```
--retoune 7
SELECT 2*3 + 1;
```

La clause suivante retourne la version de PostgreSQL

```
--PostgreSQL 15.4, compiled by Visual C++ build 1914, 64-bit
SELECT VERSION();
```

La clause FROM

La clause FROM indique la table ou les tables à partir de laquelle vous souhaitez récupérer les données.

La clause suivante retourne tous les noms et prénoms de la table élève

```
SELECT nom, prenom
FROM eleve;
```

La structure est de la forme

```
SELECT colonne1, colonne2
FROM table1;
```

La clause FROM peut récupérer les données d'une ou de plusieurs tables.

La clause FROM

Si la clause FROM récupère les données de plusieurs table, la requête renverra par défaut le produit cartésien de toutes les tables

Exemple si la table `eleve` contient les prénoms ‘Gaston’ et ‘Harry’ et que la table `matiere` contient les libellés ‘Magie’, ‘Math’, ‘Histoire’ et ‘Français’.
La requête suivante retourne 8 lignes (2x4)

```
SELECT prenom, libelle FROM matiere, eleve;
```

```
 prenom | libelle
--------+----------
 Gaston | Magie
 Harry  | Magie
 Gaston | Math
 Harry  | Math
 Gaston | Histoire
 Harry  | Histoire
 Gaston | Français
 Harry  | Français
(8 rows)
```

La clause FROM

La clause SELECT peut retourner toutes les colonnes d'une table grâce au wildcard `*`.

Exemple

```
SELECT * FROM eleve;
```

```
 id |    nom     | prenom | date_naissance
----+------------+--------+----------------
  1 | Pourquier  | Gaston | 1943-02-22
  2 | Potter     | Harry  | 1991-10-15
```

Vous pouvez renommer le nom d'une table

```
SELECT m.libelle FROM matiere AS m;
```

Le mot clé AS est facultatif dans la requête du haut.

La clause WHERE

La clause WHERE permet de spécifier les critères de filtrage pour les lignes à inclure dans le résultat. Les lignes qui ne satisfont pas la condition ne sont pas incluses.

Exemple

Dans la clause suivante, on ne récupère que les données ayant comme valeur 1 à la colonne `id`

```
SELECT * FROM eleve WHERE id=1;
```

La condition après le WHERE peut contenir plusieurs contraintes à l'aide des mots clé `AND`, `OR` et `NOT`.

```
SELECT * FROM eleve WHERE nom='Pourquier' and prenom='Gaston';
```

La clause WHERE

Dans la condition après le WHERE on peut utiliser des opérateurs de comparaison.

Voici les différents opérateurs de comparaison que l'on peut utiliser dans une clause where

<	inférieur à
>	supérieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
=	égal à
<>	différent de
!=	différent de

Vous pouvez également sélectionner une valeur entre 2 bornes à l'aide des mots clé BETWEEN et AND.
Exemple

```
WHERE note BETWEEN 5 AND 15
```

La clause GROUP BY

La clause GROUP BY est utilisée pour regrouper les lignes en fonction des valeurs de certaines colonnes. Cette clause est souvent utilisée avec des fonctions d'agrégation telles que COUNT, SUM, MIN, MAX, AVG, …

Exemple

```
SELECT count(*) AS nb, nom
FROM eleve
GROUP BY nom;
```

La clause précédente compte le nombre de lignes pour chaque nom dans la table `eleve`.

La clause HAVING

La clause GROUP BY permet de spécifier des conditions pour filtrer les groupes de lignes générés par GROUP BY. Les groupes qui ne satisfont pas la condition sont exclus du résultat.

Généralement le filtre s'effectue sur les valeurs d'agrégation.

Exemple

```
SELECT count(*) AS nb, nom
FROM eleve
GROUP BY nom
HAVING count(*) > 1;
```

La clause ORDER BY

La clause ORDER BY est utilisée pour trier les résultats en fonction des valeurs de une ou plusieurs colonnes. Vous pouvez spécifier l'ordre de tri (ascendant avec ASC ou descendant avec DESC).

Par défaut le tri s’effectue par ordre ascendant.

Vous pouvez spécifier un tri sur plusieurs colonnes

Exemple

```
SELECT * FROM eleve ORDER BY nom ASC, prenom DESC;
```

La clause LIMIT

La clause LIMIT permet de limiter le nombre de lignes renvoyées dans le résultat. C'est utile pour paginer les résultats ou pour limiter la quantité de données renvoyées.

Exemple

```
SELECT * FROM eleves LIMIT 100;
```

Pour récupérer les 100 premières lignes d'une table.

La clause LIMIT peut être associée à un OFFSET pour sauter un nombre de lignes qui nous intéressent pas.
LIMIT nombre_de_lignes_a_renvoyer OFFSET nombre_de_lignes_a_ignorer

Exemple

```
SELECT * FROM eleves LIMIT 100 OFFSET 50;
```

La commande précédente renvoie les lignes 51 à 150.

Les jointures

Vous avez vu précédemment qu'une clause FROM récupérant les données de plusieurs tables, renvoyait le produit cartésien de ces 2 tables.

Exemple

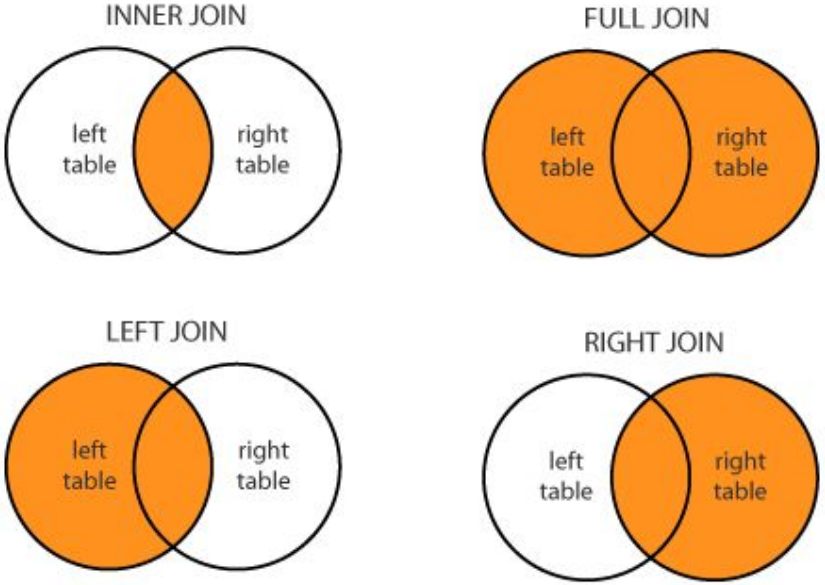
```
SELECT prenom, libelle FROM matiere, eleve;
```

Généralement lorsque nous récupérons des données de plusieurs tables, nous avons des relations entre les données des tables récupérées. Ces relations sont généralement spécifiées par des PRIMARY KEY et des FOREIGN KEY.

Les colonnes PRIMARY KEY et FOREIGN KEY vont nous permettre de joindre les données de 2 ou plusieurs tables pour ne récupérer que les données qui nous intéresse.

Les jointures

Voici les principaux types de jointures utilisés



La jointure INNER JOIN

La commande INNER JOIN (jointure interne) renvoie uniquement les lignes pour lesquelles il existe une correspondance dans les deux tables. Cela signifie que seules les lignes avec des valeurs correspondantes dans les colonnes spécifiées seront incluses dans le résultat.

Prenoms l'exemple suivant dans lequel nous possédons 2 tables

eleves (id, prenom, nom)

notes (id, note, matiere, #id_eleve)

La clé étrangères notes.id_eleve référence la colonne `id` de la table notes.

```
ALTER TABLE notes ADD FOREIGN KEY (id_eleve) REFERENCES eleves(id);
```

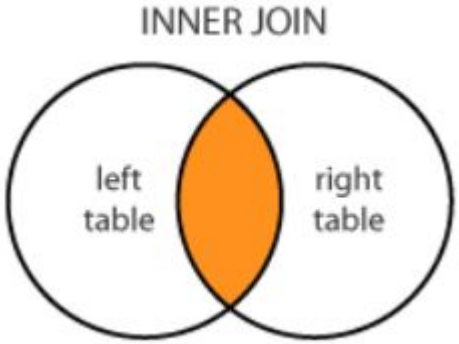
Pour récupérer les notes de l'élève Gaston Pourquier nous devons joindre les tables `eleves` et `notes` et sélectionner l'élève qui nous intéresse afin de ne pas récupérer les notes des autres élèves.

La jointure INNER JOIN

Pour ne récupérer que les notes de l'élève "Gaston Pourquier".

Nous pouvons utiliser la requête suivante

```
SELECT n.matiere, n.note
FROM eleves e
INNER JOIN notes n ON n.id_eleve = e.id
WHERE e.prenom='Gaston' AND e.nom='Pourquier';
```



La jointure INNER JOIN

Autre exemple avec les tables Utilisateurs et Commandes.

Utilisateurs (id, nom, prenom)
Commandes (id, ref_commande, #utilisateur_id)

Où la colonne `utilisateur_id` de la table `Commandes` référence la colonne `id` de la table utilisateur.

Pour avoir la référence des commandes pour chaque utilisateur, nous pouvons exécuter la commande suivante

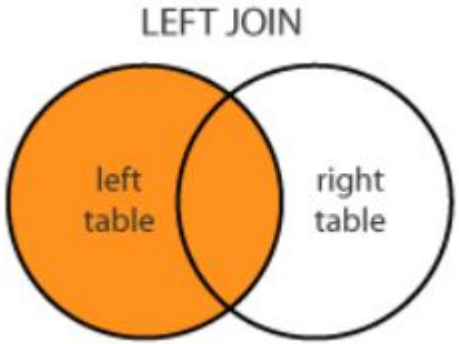
```
SELECT u.nom, u.prenom, c.ref_commande
FROM Utilisateurs u
INNER JOIN Commandes c ON u.id = c.utilisateur_id;
```

La jointure LEFT JOIN

La jointure LEFT JOIN (Jointure à Gauche) renvoie toutes les lignes de la table de gauche (première table spécifiée) et les lignes correspondantes de la table de droite (deuxième table spécifiée). Si aucune correspondance n'est trouvée dans la table de droite, des valeurs NULL sont renvoyées pour les colonnes de la table de droite.

Exemple: nous voulons obtenir la liste de tous les utilisateurs et, le cas échéant, leurs commandes associées.

```
SELECT u.nom, u.prenom, c.ref_commande
FROM Utilisateurs u
LEFT JOIN Commandes c ON u.id = c.utilisateur_id;
```

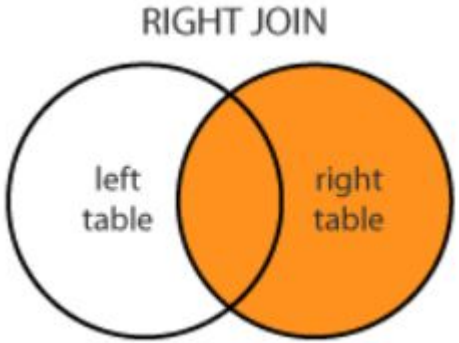


La jointure RIGHT JOIN

RIGHT JOIN (Jointure à Droite) est similaire à une jointure à gauche, mais elle renvoie toutes les lignes de la table de droite et les lignes correspondantes de la table de gauche. Si aucune correspondance n'est trouvée dans la table de gauche, des valeurs NULL sont renvoyées pour les colonnes de la table de gauche.

Exemple: nous voulons obtenir la liste de toutes les commandes et, le cas échéant, les noms des utilisateurs correspondants.

```
SELECT u.nom, u.prenom, c.ref_commande
FROM Utilisateurs u
RIGHT JOIN Commandes c ON u.id = c.utilisateur_id;
```

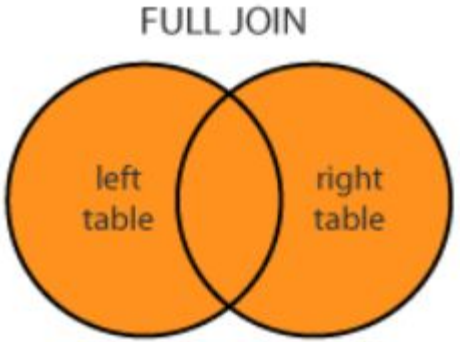


La jointure FULL JOIN

La jointure FULL JOIN ou FULL OUTER JOIN (Jointure complète) renvoie toutes les lignes des deux tables, en combinant les lignes correspondantes et en renvoyant des valeurs NULL lorsque des correspondances ne sont pas trouvées.

Exemple: nous voulons obtenir la liste de tous les utilisateurs et de toutes les commandes, en montrant les correspondances entre les deux.

```
SELECT u.nom, u.prenom, c.ref_commande
FROM Utilisateurs u
FULL JOIN Commandes c ON u.id = c.utilisateur_id;
```



Les fonctions d’agrégations

En SQL, les fonctions d’agrégation permettent de réaliser des opérations arithmétiques et statistiques au sein d'une requête.

Les principales fonctions sont les suivantes

COUNT() pour compter le nombre d’enregistrements d'une table ou d'une colonne distincte.

AVG() pour calculer la moyenne sur un ensemble d’enregistrements.

SUM() pour calculer la somme sur un ensemble d’enregistrements.

MAX() pour récupérer la valeur maximum d’une colonne sur un ensemble d’enregistrements.

MIN() pour récupérer la valeur minimum d’une colonne sur un ensemble d’enregistrements.

Les fonctions d’agrégations

Pour retourner le nombre de colonnes d’une table, vous pouvez utiliser la requête suivante

```
SELECT count(*) AS nb_colonnes FROM notes;
```

Pour retourner le nombre de valeurs non nulles d’un champ spécifique, vous pouvez utiliser la commande suivante

```
SELECT count(nom_du_champ) AS nb FROM nom_table;
```

Exemple

```
SELECT count(note) AS nb_colonnes FROM notes;
```

Vous pouvez également utiliser les autre fonctions d'agrégation

```
SELECT min(note) AS min_note, max(note) AS max_note, avg(note) AS avg_note FROM notes;
```

Les fonctions d’agrégations

Attention si dans une clause SELECT vous mélanger des fonctions d’agrégation (COUNT, SUM, MIN, MAX, AVG) avec d’autres champs, vous devez utiliser ces autres champs dans une clause GROUP BY.

Autrement vous aurez un message d’erreur.

Exemple

```
SELECT count(*), nom
FROM eleve
GROUP BY nom;
```