Module BASES DE DONNEES RELATIONNELLES
TD 2 2e partie
SQL

Hervé Owsinski
Herve.Owsinski@uphf.fr

2026-2027

1 GROUP BY : les regoupements

Exemple de l'effet d'un GROUP BY

Explications : regardons ce que sont les employés :

```
MariaDB [TD4]> SELECT * FROM Employe;
+-----+----------+----------+------------+------------+--------------+--------+
| idE | nom      | prenom   | fonction   | dateEmb    | ville        | idDept |
+-----+----------+----------+------------+------------+--------------+--------+
|   1 | Muller   | Gerard   | ingenieur  | 1995-02-01 | Valenciennes |      1 |
|   2 | Rochat   | Juliette | ingenieur  | 1992-03-04 | Lille        |      1 |
|   3 | Legrand  | Didier   | technicien | 1995-02-01 | Paris        |      2 |
|   4 | Lepetit  | Jean     | technicien | 1992-03-04 | Maubeuge     |      1 |
|   5 | Chan     | Jean     | technicien | 1992-03-04 | Lille        |      4 |
|   6 | Mini     | Jeanne   | vendeur    | 1992-03-04 | Geneve       |      3 |
|   7 | Durand   | Paul     | ingenieur  | 2000-01-10 | Lyon         |      1 |
|   8 | Fontaine | Claire   | technicien | 2001-02-12 | Marseille    |      2 |
|   9 | Girard   | Luc      | vendeur    | 2002-03-14 | Nice         |      3 |
|  10 | Noel     | Sarah    | ingenieur  | 2003-04-16 | Paris        |      1 |
|  11 | Petit    | Thomas   | technicien | 2004-05-18 | Lille        |      4 |
|  12 | Lemoine  | Emma     | vendeur    | 2005-06-20 | Geneve       |      2 |
|  13 | Blanc    | Marie    | ingenieur  | 2006-07-22 | Lyon         |      3 |
|  14 | Rousseau | Julien   | technicien | 2007-08-24 | Paris        |      4 |
|  15 | Garnier  | Lucie    | vendeur    | 2008-09-26 | Nice         |      1 |
+-----+----------+----------+------------+------------+--------------+--------+
15 rows in set (0,001 sec)
```

Exemple de l'effet d'un GROUP BY

On regroupe un peu les employés en les triant selon idDept :

```
MariaDB [TD4]> SELECT * FROM Employe ORDER BY idDept ASC, idE ASC;
+-----+----------+----------+------------+------------+--------------+--------+
| idE | nom      | prenom   | fonction   | dateEmb    | ville        | idDept |
+-----+----------+----------+------------+------------+--------------+--------+
|   1 | Muller   | Gerard   | ingenieur  | 1995-02-01 | Valenciennes |      1 |
|   2 | Rochat   | Juliette | ingenieur  | 1992-03-04 | Lille        |      1 |
|   4 | Lepetit  | Jean     | technicien | 1992-03-04 | Maubeuge     |      1 |
|   7 | Durand   | Paul     | ingenieur  | 2000-01-10 | Lyon         |      1 |
|  10 | Noel     | Sarah    | ingenieur  | 2003-04-16 | Paris        |      1 |
|  15 | Garnier  | Lucie    | vendeur    | 2008-09-26 | Nice         |      1 |
|   3 | Legrand  | Didier   | technicien | 1995-02-01 | Paris        |      2 |
|   8 | Fontaine | Claire   | technicien | 2001-02-12 | Marseille    |      2 |
|  12 | Lemoine  | Emma     | vendeur    | 2005-06-20 | Geneve       |      2 |
|   6 | Mini     | Jeanne   | vendeur    | 1992-03-04 | Geneve       |      3 |
|   9 | Girard   | Luc      | vendeur    | 2002-03-14 | Nice         |      3 |
|  13 | Blanc    | Marie    | ingenieur  | 2006-07-22 | Lyon         |      3 |
|   5 | Chan     | Jean     | technicien | 1992-03-04 | Lille        |      4 |
|  11 | Petit    | Thomas   | technicien | 2004-05-18 | Lille        |      4 |
|  14 | Rousseau | Julien   | technicien | 2007-08-24 | Paris        |      4 |
+-----+----------+----------+------------+------------+--------------+--------+
15 rows in set (0,001 sec)
```

Exemple de l'effet d'un group by

Le GROUP BY récupère les valeurs distinctes sur idDept et va générer plusieurs sous-tables :

```
+-----+----------+----------+------------+------------+--------------+--------+
| idE | nom      | prenom   | fonction   | dateEmb    | ville        | idDept |
+-----+----------+----------+------------+------------+--------------+--------+
|   1 | Muller   | Gerard   | ingenieur  | 1995-02-01 | Valenciennes |      1 |
|   2 | Rochat   | Juliette | ingenieur  | 1992-03-04 | Lille        |      1 |
|   4 | Lepetit  | Jean     | technicien | 1992-03-04 | Maubeuge     |      1 |
```

```
|   7 | Durand   | Paul     | ingenieur  | 2000-01-10 | Lyon         |      1 |
|  10 | Noel     | Sarah    | ingenieur  | 2003-04-16 | Paris        |      1 |
|  15 | Garnier  | Lucie    | vendeur    | 2008-09-26 | Nice         |      1 |
+-----+----------+----------+------------+------------+--------------+--------+
|   3 | Legrand  | Didier   | technicien | 1995-02-01 | Paris        |      2 |
|   8 | Fontaine | Claire   | technicien | 2001-02-12 | Marseille    |      2 |
|  12 | Lemoine  | Emma     | vendeur    | 2005-06-20 | Geneve       |      2 |
+-----+----------+----------+------------+------------+--------------+--------+
|   6 | Mini     | Jeanne   | vendeur    | 1992-03-04 | Geneve       |      3 |
|   9 | Girard   | Luc      | vendeur    | 2002-03-14 | Nice         |      3 |
|  13 | Blanc    | Marie    | ingenieur  | 2006-07-22 | Lyon         |      3 |
+-----+----------+----------+------------+------------+--------------+--------+
|   5 | Chan     | Jean     | technicien | 1992-03-04 | Lille        |      4 |
|  11 | Petit    | Thomas   | technicien | 2004-05-18 | Lille        |      4 |
|  14 | Rousseau | Julien   | technicien | 2007-08-24 | Paris        |      4 |
+-----+----------+----------+------------+------------+--------------+--------+
15 rows in set (0,001 sec)
```

16. Afficher, pour chaque client, le nombre de commandes qu'il a passées (les clients sans commande doivent apparaître avec 0).

```sql
SELECT Cl.id_client, Cl.nom, Cl.prenom,
       COUNT(Co.id_commande) AS nombre_commandes
FROM Client Cl
LEFT JOIN Commande Co ON Co.id_client = Cl.id_client
GROUP BY Cl.id_client, Cl.nom, Cl.prenom;
```

Ici, on regroupe les lignes obtenues par client.
COUNT n'agit plus sur toute la table mais sur chacune des sous-tables : une par client. nombre_commandes correspond donc au nombre de commande de chaque client.

16. Afficher, pour chaque client, le nombre de commandes qu'il a passées (les clients sans commande doivent apparaître avec 0).

```sql
SELECT Cl.id_client, Cl.nom, Cl.prenom,
       COUNT(Co.id_commande) AS nombre_commandes
FROM Client Cl
LEFT JOIN Commande Co USING (id_client)
GROUP BY Cl.id_client, Cl.nom, Cl.prenom;
```

Attention à USING : les deux attributs doivent avoir le même nom. C'est courant mais si vous ne connaissez que la méthode avec USING, vous ne parviendrez pas à gérer un cas nécessitant ON A.idA = B.idB.
Différence avec ON ? La colonne commune n'apparaît qu'une seule fois et pas deux.

17. Afficher le chiffre d'affaires réalisé par catégorie de produits, de la plus rentable à la moins rentable

```sql
SELECT P.categorie,
       SUM(L.quantite * P.prix) AS chiffre_affaires
FROM Produit P
JOIN Ligne_commande L ON L.id_produit = P.id_produit
GROUP BY P.categorie
ORDER BY chiffre_affaires DESC;
```

On regroupe les lignes de commandes par catégorie de produts. On obtient donc une sous-table pour chacune des sous-tables.
Le SUM fait donc la somme de tous les chiffres d'affaires d'une catégorie à la fois.
Remarquez qu'on peut utiliser un résultat calculé dans SELECT dans le ORDER BY.

17. Afficher le chiffre d'affaires réalisé par catégorie de produits, de la plus rentable à la moins rentable

```sql
SELECT P.categorie,
       SUM(L.quantite * P.prix) AS chiffre_affaires
FROM Produit P
JOIN Ligne_commande L ON L.id_produit = P.id_produit
JOIN Commande C       ON C.id_commande = L.id_commande
```

```sql
WHERE C.statut = 'payée'
GROUP BY P.categorie
ORDER BY chiffre_affaires DESC;
```

On regroupe les lignes de commandes par catégorie de produts. On obtient donc une sous-table pour chacune des sous-tables.
Le SUM fait donc la somme de tous les chiffres d'affaires d'une catégorie à la fois.
Remarquez qu'on peut utiliser un résultat calculé dans SELECT dans le ORDER BY.

17. Afficher le chiffre d'affaires réalisé par catégorie de produits, de la plus rentable à la moins rentable

```sql
SELECT P.categorie,
       SUM(L.quantite * P.prix) AS chiffre_affaires
FROM Produit P
JOIN Ligne_commande L USING (id_produit)
JOIN Commande C       USING (id_commande)
WHERE C.statut = 'payée'
GROUP BY P.categorie
ORDER BY chiffre_affaires DESC;
```

18. Afficher le total dépensé par chaque client, en ignorant les commandes annulées.

```sql
SELECT Cl.id_client, Cl.nom, Cl.prenom,
       COALESCE(SUM(L.quantite * P.prix), 0) AS total_depense
FROM Client Cl
LEFT JOIN Commande Co ON Co.id_client = Cl.id_client
                      AND Co.statut <> 'annulée'
LEFT JOIN Ligne_commande L ON L.id_commande = Co.id_commande
LEFT JOIN Produit P        ON P.id_produit = L.id_produit
GROUP BY Cl.id_client, Cl.nom, Cl.prenom ;
```

On remarque que le JOIN entre Client et Commande se fait uniquement si la commande n'est pas annulée. En conséquence, les clients n'ayant que des commandes annulées sont présents néanmoins.

19. Reprendre la question précédente en ne gardant que les clients qui ont dépensé plus de 100 €.

```sql
SELECT Cl.id_client, Cl.nom, Cl.prenom,
       SUM(L.quantite * P.prix) AS total_depense
FROM Client Cl
JOIN Commande Co      ON Co.id_client = Cl.id_client
JOIN Ligne_commande L ON L.id_commande = Co.id_commande
JOIN Produit P        ON P.id_produit = L.id_produit
WHERE Co.statut <> 'annulée'
GROUP BY Cl.id_client, Cl.nom, Cl.prenom
HAVING SUM(L.quantite * P.prix) > 100;
```

WHERE sélectionne les lignes à garder : il agit sur les lignes.
GROUP BY regroupe les lignes restantes en sous-groupes.
HAVING sélectionne les sous-groupes à garder : il agir sur les regroupements.

19. Reprendre la question précédente en ne gardant que les clients qui ont dépensé plus de 100 €.

```sql
SELECT Cl.id_client, Cl.nom, Cl.prenom,
       SUM(L.quantite * P.prix) AS total_depense
FROM Client Cl
JOIN Commande Co      USING (id_client)
JOIN Ligne_commande L USING (id_commande)
JOIN Produit P        USING (id_produit)
WHERE Co.statut <> 'annulée'
GROUP BY Cl.id_client, Cl.nom, Cl.prenom
HAVING SUM(L.quantite * P.prix) > 100;
```

2 sous-requête (ou pas)

20. Afficher les produits dont le prix est supérieur au prix moyen de tous les produits.

```sql
SELECT libelle
FROM Produit
WHERE prix > (SELECT AVG(prix) FROM Produit);
```

WHERE sélectionne les lignes à garder : cette clause travaille donc ligne par ligne et ne permet pas de comparer les lignes entre elles directement.

La solution est de récupérer d'abord la valeur moyenne via une sous-requête puis d'utiliser cette valeur dans la requête principale.

21. Afficher le nom des clients qui ont commandé au moins un produit de la catégorie « Audio ».

```sql
SELECT id_client, nom
FROM Client
WHERE id_client IN (
    SELECT CO.id_client
    FROM Produit  P
    JOIN Ligne_commande LC ON P.id_produit = LC.id_produit
    JOIN Commande CO       ON LC.id_commande = CO.id_commande
    WHERE P.categorie = 'Audio'
    );
```

21. Afficher le nom des clients qui ont commandé au moins un produit de la catégorie « Audio ».

```sql
SELECT id_client, nom
FROM Client
WHERE EXISTS (
    SELECT 1
    FROM Produit  P
    JOIN Ligne_commande LC ON P.id_produit = LC.id_produit
    JOIN Commande CO       ON LC.id_commande = CO.id_commande
    WHERE P.categorie = 'Audio'
      AND Client.id_client = CO.id_client
    );
```

21. Afficher le nom des clients qui ont commandé au moins un produit de la catégorie « Audio ».

```sql
SELECT DISTINCT C.id_client, C.nom
FROM Produit  P
JOIN Ligne_commande LC ON P.id_produit = LC.id_produit
JOIN Commande CO       ON LC.id_commande = CO.id_commande
JOIN Client C          ON CO.id_client = C.id_client
WHERE P.categorie = 'Audio';
```

```sql
SELECT DISTINCT C.id_client, C.nom
FROM Produit  P
JOIN Ligne_commande LC USING (id_produit)
JOIN Commande CO       USING (id_commande)
JOIN Client C          USING (id_client)
WHERE P.categorie = 'Audio';
```

22. Afficher les produits qui n'ont jamais été commandés.

```sql
SELECT libelle
FROM Produit
WHERE id_produit NOT IN (
    SELECT id_produit
    FROM Ligne_commande
    );
```

22. Afficher les produits qui n'ont jamais été commandés.

```sql
SELECT libelle
FROM Produit
WHERE NOT EXISTS (
    SELECT 1
    FROM Ligne_commande
    WHERE Ligne_commande.id_produit = Produit.id_produit
);
```

22. Afficher les produits qui n'ont jamais été commandés.

```sql
SELECT libelle
FROM Produit P
LEFT JOIN Ligne_commande LC ON P.id_produit = LC.id_produit
WHERE LC.id_produit IS NULL;
```

```sql
SELECT libelle
FROM Produit P
LEFT JOIN Ligne_commande LC USING (id_produit)
WHERE LC.id_produit IS NULL;
```

22. Afficher les produits qui n'ont jamais été commandés.

```sql
SELECT libelle
FROM Produit P
EXCEPT
SELECT libelle
FROM Produit P
JOIN Ligne_commande LC ON P.id_produit = LC.id_produit;
```

23. Afficher le nom des clients qui ont commandé au moins un produit de chaque catégorie.

```sql
SELECT C.nom
FROM Client C
JOIN Commande CO       ON C.id_client = CO.id_client
JOIN Ligne_commande LC ON CO.id_produit = LC.id_produit
JOIN produit P         ON LC.id_produit = P.id_produit
GROUP BY C.id_client, C.nom
HAVING COUNT(DISTINCT categorie) = (
    SELECT COUNT(DISTINCT categorie) FROM produit
    );
```

3 Opérateurs ensemblistes (ou pas)

24. Produire une liste unique contenant à la fois les villes des clients et les catégories de produits.

```sql
SELECT ville AS texte
FROM Client
UNION
SELECT categorie
FROM Produit;
```

25. Afficher les clients qui ont commandé à la fois un produit 'Informatique' et un produit 'Audio'.

```sql
SELECT CO.id_client
FROM Commande CO
JOIN Ligne_commande LC ON CO.id_commande = LC.id_commande
```

```sql
JOIN Produit P         ON LC.id_produit  = P.id_produit
WHERE categorie = 'Informatique'
INTERSECT
SELECT CO.id_client
FROM Commande
JOIN Ligne_commande LC ON CO.id_commande = LC.id_commande
JOIN Produit P         ON LC.id_produit = P.id_produit
WHERE categorie = 'Audio';
```

25. Afficher les clients qui ont commandé à la fois un produit 'Informatique' et un produit 'Audio'.

```sql
SELECT DISTINCT CO.id_client
FROM Commande CO
JOIN Ligne_commande LC ON CO.id_commande = LC.id_commande
JOIN Produit P         ON LC.id_produit  = P.id_produit
WHERE categorie = 'Informatique'
  AND CO.id_client IN (
         SELECT CO.id_client
         FROM Commande
         JOIN Ligne_commande LC ON CO.id_commande = LC.id_commande
         JOIN Produit P         ON LC.id_produit = P.id_produit
         WHERE categorie = 'Audio'
      );
```

26. Afficher les clients qui n'ont jamais passé de commande, en utilisant un opérateur ensembliste.

```sql
SELECT id_client
FROM Client
EXCEPT
SELECT id_client
FROM Commande;
```

26. Afficher les clients qui n'ont jamais passé de commande, en utilisant un opérateur ensembliste OU PAS.

```sql
SELECT id_client
FROM Client
WHERE id_client NOT IN (
    SELECT id_client
    FROM Commande
    );
```

```sql
SELECT id_client
FROM Client
WHERE NOT EXISTS (
    SELECT 1
    FROM Commande
    WHERE Client.id_client = Commande.id_client
    );
```

4 Les vues

27. Créer une vue `v_detail_commande` qui regroupe, pour chaque ligne de commande : le numéro de commande, la date, le statut, l'identifiant et le nom du client, sa ville, le libellé et la catégorie du produit, la quantité et le total de la ligne.

```sql
-- CREATE VIEW en erreur si la vue existe déjà donc...
DROP VIEW IF EXISTS v_detail_commande ;

CREATE VIEW v_detail_commande AS
SELECT Commande.*, nom, ville, libelle, categorie, quantite,
```

```
       quantite * prix AS total_ligne
FROM Commande
JOIN Client c USING (id_client)
JOIN Ligne_commande USING (id_commande)
JOIN Produit USING (id_produit);
```

28. À l'aide de cette vue, afficher toutes les lignes de commande des clients habitant à Lille.

```
CREATE VIEW v_detail_commande AS
SELECT Commande.*, nom, ville, libelle, categorie, quantite,
       quantite * prix AS total_ligne
...
```

donc :

```
SELECT * FROM v_detail_commande
WHERE ville='Lille';
```

29. Créer une vue `v_bilan_client` construite à partir de la vue précédente, qui donne pour chaque client le nombre de commandes et le total dépensé (hors commandes annulées).

```
CREATE VIEW v_detail_commande AS
SELECT Commande.*, nom, ville, libelle, categorie, quantite,
       quantite * prix AS total_ligne
...
```

donc :

```
DROP VIEW IF EXISTS v_detail_client ;

CREATE VIEW v_bilan_client  (nom, nb, total) AS
SELECT nom, count(DISTINCT id_commande), SUM(total_ligne)
FROM v_detail_commande
WHERE statut <> 'annulé'    -- ou !=
GROUP BY id_client, nom;
```

30. l'aide de cette dernière vue, afficher les clients dont le total dépensé est supérieur à la moyenne, du plus dépensier au moins dépensier.

```
CREATE VIEW v_bilan_client  (nom, nb, total) AS
...
```

donc :

```
SELECT nom, total
FROM v_bilan_client
WHERE total > (SELECT AVG(total) FROM v_bilan_client)
ORDER BY total desc ;
```