SGBD - 2e

Chapitre 4 - Langage de Manipulation des Données (LMD)

Daniel Schreurs

10 octobre 2022

Haute École de la Province de Liège

Table des matières du chapitre i

Table des matières du chapitre ii

Introduction

Table des matières de la section : Introduction i

Table des matières de la section : Introduction ii

Introduction : Définition

Un langage de manipulation des données, ou LMD, contient des commandes :

- d'interrogation (recherche)
- de modification.

Toutes ces commandes concernent toujours un ensemble de tuples qui satisfait un critère de qualification.

Introduction : Définition

En SQL :

- Commande d'interrogation : SELECT
- Commandes de modification :
  - ajout (INSERT),
  - mise à jour (UPDATE)
  - suppression (DELETE)

Remarque : dans ce chapitre, la présentation des commandes de modification se fera sans tenir compte du concept de transaction (accès concurrents) qui sera abordé dans le chapitre 5.

Introduction : Remarques i

Quelques remarques à propos de l'écriture des requêtes SQL :

- SQL est insensible à la casse
- La syntaxe des commentaires est :
  - `--` : commentaire sur une seule ligne
  - `/* ... */` : commentaire sur plusieurs lignes
- Les chaînes de caractères sont écrites entre simple quote, une apostrophe dans une chaîne est dédoublée
- Ex : `'Il fait beau aujourd''hui'`
- Si un élément de la base a pour nom un mot réservé, il faut le spécifier entre guillemets Ex : `CREATE TABLE "TABLE"( ... );`
- Les noms des objets
  - ont une longueur maximale de 128 caractères [1].

Introduction : Remarques ii

  - doivent commencer par une lettre, peuvent contenir les caractères a à z, 0 à 9, $, # et _

- L'opérateur AS sert à donner un nom à une colonne sélectionnée ou calculée dans une requête :

```
SELECT COUNT(*)AS Nbre FROM Employes;
```

1. Pour plus d'informations, consultez la documentation.

Recherche de base

Table des matières de la section : Recherche de base i

Table des matières de la section : Recherche de base ii

Recherche de base : SELECT

Clause SELECT

```
select_de_base ::=
        SELECT [ ALL | DISTINCT ] clause_de_sélection
        FROM nom_table
        [ WHERE condition ]
        ;

        clause_de_sélection ::=
        * |  nom_table.*  |  liste_colonne
```

Recherche de base : WHERE

La clause WHERE permet de spécifier un critère de sélection.

Clause WHERE

```
condition ::=
    [ NOT ] condition_de_base
  | condition_between
  | condition_in
  | condition_like
  | condition_null
  | condition AND | OR condition
  | ( condition )
```

Recherche de base : WHERE

Les possibilités de la clause WHERE sont étendues en développant la syntaxe de **`condition_de_base`**.

Condition de base

```
condition_de_base ::=
    expression oper_comp expression

oper_comp ::= = | <> | < | <= | > | >=

expression ::= expression_numérique
   | expression_caractère
   | expression_date_temps
   | expression_intervalle
```

Recherche de base : WHERE

SQL2 va plus loin dans la définition de la notion de **`condition_de_base`** qui est appelée dans la norme comparison expression.

comparison expression

```
row_constructor comparison_operator row_constructor

comparison_operator ::=     = | <> | < | <= | > | >=

row_constructor ::= liste de valeurs
 | ( table_expression )
```

2

2. Une **`table_expression`** est une expression dont le résultat doit contenir au plus une ligne (selon la norme, une table ne contenant pas de lignes est convertie en une table contenant une ligne dont tous les éléments sont NULL).

Recherche de base : WHERE

Exemples de row_constructor

```sql
SELECT Nom
FROM Employes
WHERE bareme >
      (SELECT bareme
       FROM Employes
       WHERE nom = 'BEART');

SELECT Nom
FROM Employes
WHERE (sexe, bareme, numdep) =
      (SELECT DISTINCT 'F', 80000, 'd00001'
       FROM Employes);
```

Recherche de base : LIKE

L'opérateur LIKE permet de comparer des chaînes de caractères

```
nom_col [NOT] LIKE
  'modèle de chaîne'
  [ESCAPE escape-car]
```

- `_` remplace un seul caractère
- `%` remplace un nombre quelconque (éventuellement `NULL`) de caractères

Recherche de base : LIKE

Exemple : obtenir le nom des vues qui commencent par USER_

```
1 SELECT object_name
2 FROM all_objects
3 WHERE object_name LIKE 'USER@_%' ESCAPE '@'
4     AND object_type = 'VIEW';
```

Recherche de base : LIKE

La comparaison tient compte de la longueur du type de la donnée

```sql
CREATE TABLE Chaines
(
    NomChar4      CHAR(4),
    NomChar10     CHAR(10),
    NomVarChar4   VARCHAR(4),
    NomVarChar10 VARCHAR(10)
);

INSERT INTO Chaines
VALUES ('SQL', 'SQL', 'SQL', 'SQL');
COMMIT;

SELECT *
FROM Chaines
WHERE NomChar4 LIKE NomChar10;
```

- Aucun résultat !!!

Recherche de base : LIKE

La comparaison tient compte de la longueur du type de la donnée

```sql
CREATE TABLE Chaines
(
    NomChar4      CHAR(4),
    NomChar10     CHAR(10),
    NomVarChar4   VARCHAR(4),
    NomVarChar10 VARCHAR(10)
);

INSERT INTO Chaines
VALUES ('SQL', 'SQL', 'SQL', 'SQL');
COMMIT;

SELECT *
FROM Chaines
WHERE NomVarChar4 LIKE NomVarChar10;
```

- On obtient un résultat !

Recherche de base : NULL

- `NULL` est une absence de valeur
- Le résultat de la comparaison (`<, <=, =, >=, >, <>`) entre `NULL` et une autre valeur (même `NULL`) n'est ni **VRAI** ni faux, mais inconnu Ex : `NULL = NULL` ou `NULL < NULL` vaut inconnu

Recherche de base : NULL

AND	V	I	F
V	V	I	F
I	I	I	F
F	F	F	F

OR	V	I	F
V	V	V	V
I	V	I	I
F	V	I	F

	NOT
V	F
I	I
F	V

Figure 1 – Tables de vérité : Vrai, faux, inconnu

Recherche de base : NULL

Deux opérateurs spéciaux pour comparer une valeur à NULL

```sql
SELECT nom
FROM employes
WHERE numchef IS NULL;

SELECT nom
FROM employes
WHERE numchef is not NULL;
```

Recherche de base : NULL

COALESCE recherche la première valeur non NULL dans une liste de valeurs

```sql
SELECT nom,
       COALESCE(numchef, 'Président') AS Chef
FROM employes;
```

Recherche de base : NULL

Tester si une valeur a été modifiée

```
COALESCE (OldValue, 'XXX') =
  COALESCE (CurrentValue, 'XXX')
```

Recherche de base : Union

Union, Intersection, différence : Les relations doivent être de même degré et les colonnes correspondantes doivent être compatibles

Recherche de base : Union

Union

```
A  UNION [ALL]  B
   [CORRESPONDING [BY (liste_colonne)]]
```

`liste_colonne` : $C_1, C_2, ..., C_n$! Chaque C_i doit être présente à la fois dans les tables A et B

Recherche de base : Union

Exemple : rechercher le nom et le prénom des employées qui habitent dans la commune de Liège ou Waremme

```sql
SELECT nom, prenom
FROM Employes
WHERE UPPER(commune) = 'LIEGE'
  and sexe = 'F'
UNION
SELECT nom, prenom
FROM Employes
WHERE UPPER(commune) = 'WAREMME'
  and sexe = 'F';
```

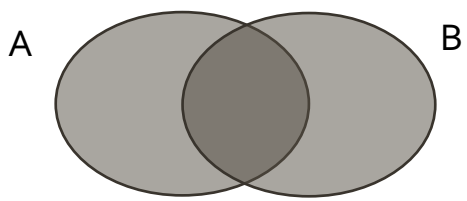

Figure 2 – union

Recherche de base : EXCEPT

Différence

```
A  EXCEPT [ALL]   B
   [CORRESPONDING [BY (liste_colonne)]]
```

liste_colonne : $C_1, C_2, ..., C_n$! Chaque C_i doit être présente à la fois dans les tables A et B

Recherche de base : EXCEPT

Exemple : rechercher le numéro des employés qui n'ont pas de projets en cours

```sql
SELECT NumSecu
FROM Employes MINUS -- en Oracle 10g
SELECT NumSecu
FROM EmpPro;
```

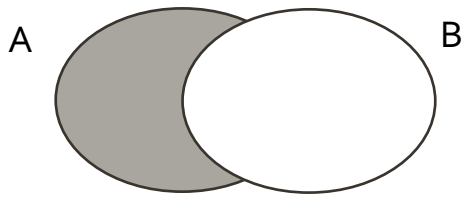

Figure 3 – difference

Recherche de base : INTERSECT

Différence

```
A  INTERSECT [ALL]  B
   [CORRESPONDING [BY (liste_colonne)]]
```

liste_colonne : $C_1, C_2, ..., C_n$! Chaque C_i doit être présente à la fois dans les tables *A* et *B*

Recherche de base : INTERSECT

Exemple : rechercher le numéro des employés qui travaillent sur les projets p10346 et p10349

```sql
SELECT NumSecu
FROM EmpPro
WHERE Numpro = 'p10346'
INTERSECT
SELECT NumSecu
FROM EmpPro
WHERE Numpro = 'p10349';
```

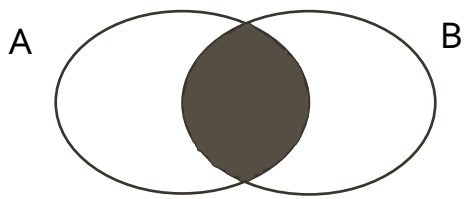

Figure 4 – intersection

Recherche de base : INTERSECT

$$A \cap B = A - (A - B) = B - (B - A)$$

Exemple : rechercher le numéro des employés qui travaillent sur les projets p10346 et p10349

```
1 SELECT NumSecu
2 FROM EmpPro
3 WHERE Numpro = 'p10346' MINUS
4     (SELECT NumSecu FROM EmpPro
5      WHERE Numpro = 'p10346'
6      MINUS
7      SELECT NumSecu FROM EmpPro
8      WHERE Numpro = 'p10349');
```

Recherche de base : INTERSECT

Requêtes erronées

```sql
-- pas de résultat : un même champ ne peut avoir 2
    valeurs différentes dans un même tuple
SELECT NumSecu
FROM EmpPro
WHERE Numpro = 'p10346'
  AND NumPro = 'p10349';

-- cela représente l'union et non l'intersection !
SELECT NumSecu
FROM EmpPro
WHERE Numpro IN ('p10346', 'p10349');
```

Recherche de base : INTERSECT

Exemple : Afficher le nom et le prénom des employés qui travaillent sur le projet p10346 et sur le projet p10349

```
1  SELECT *
2  from (SELECT NOM, PRENOM
3        FROM EmpPro
4                inner join EMPLOYES E2 on E2.NUMSECU =
                     EMPPRO.NUMSECU
5        WHERE Numpro = 'p10346'
6        INTERSECT
7        SELECT NOM, PRENOM
8        FROM EmpPro
9                inner join EMPLOYES E on E.NUMSECU =
                     EMPPRO.NUMSECU
10       WHERE Numpro = 'p10349');
```

Recherche de base : Exercices

- Afficher le numéro des départements composés uniquement de femmes, sachant que le numéro de département peut-être inconnu
- Afficher le numéro des départements composés d'hommes et de femmes, sachant que le numéro de département peut-être inconnu
- Afficher le numéro des départements composés d'employés habitant à Liège ou Herstal
- Afficher le numéro des employés qui ne dirigent personne, sachant qu'un chef de département doit être chef d'au moins un employé

Recherche de base : Exercices

Afficher le numéro des départements composés uniquement de femmes sachant que le numéro de département peut-être inconnu

```sql
SELECT COALESCE(NumDep, 'Département inconnu')
FROM Employes
WHERE UPPER(sexe) = 'F'
    MINUS
SELECT COALESCE(NumDep, 'Département inconnu')
FROM Employes
WHERE UPPER(sexe) = 'M';
```

Recherche de base : Exercices

Afficher le numéro des départements composés d'hommes et de femmes sachant que le numéro de département peut-être inconnu

```sql
SELECT COALESCE (NumDep, 'Département inconnu')
FROM Employes
WHERE UPPER(sexe) = 'F'
INTERSECT
SELECT COALESCE (NumDep, 'Département inconnu')
FROM Employes
WHERE UPPER(sexe) = 'M';
```

Recherche de base : Exercices

Afficher le numéro des départements composés d'employés habitant à Liège ou Herstal

```sql
SELECT NumDep
FROM Employes
WHERE UPPER(Commune) = 'LIEGE'
UNION
SELECT NumDep
FROM Employes
WHERE UPPER(Commune) = 'HERSTAL';
```

Recherche de base : Exercices

Afficher le numéro des employés qui ne dirigent personne sachant qu'un chef de département doit être chef d'au moins un employé

```sql
SELECT NumSecu
FROM Employes
MINUS
SELECT COALESCE (NumChef, 'XXX')
FROM Employes;
```

Recherche de base avec jointure

Table des matières de la section : Recherche de base avec jointure i

Table des matières de la section : Recherche de base avec jointure ii

Table des matières de la section : Recherche de base avec jointure iii

Recherche de base avec jointure : Jointure prédicative ou jointure manuelle

Jointure prédicative ou jointure manuelle

```sql
SELECT Nom, NomPro
FROM Employes,
     EmpPro,
     Projets
WHERE Employes.NumSecu = EmpPro.NumSecu
  AND EmpPro.NumPro = Projets.NumPro;
```

Recherche de base avec jointure : Jointure prédicative ou jointure manuelle

Jointure prédicative ou jointure manuelle

```sql
SELECT Nom, NomPro, NomDep
FROM Employes,
     EmpPro,
     Projets,
     Departements
WHERE Employes.NumSecu = EmpPro.NumSecu
  AND EmpPro.NumPro = Projets.NumPro
  AND Employes.NumDep = Departements.NumDep;
```

3

3. 4 tables impliquent 3 instructions de jointure manuelle !!!

Recherche de base avec jointure : Jointure interne non-equijointure

Une jointure avec une condition autre que l'égalité. Les conditions peuvent utiliser : `>= < <= <> IN LIKE BETWEEN EXISTS`

Donnez le nom des employés qui ont un salaire > à "BEART"

```sql
SELECT E2.Nom
FROM Employes E1
         JOIN Employes E2
              ON E2.Bareme > E1.Bareme
WHERE E1.Nom = 'BEART';
```

Recherche de base avec jointure : Auto-jointure

Afficher le nom de l'employé ainsi que le nom de son chef direct

```sql
SELECT E1.Nom, ' a pour chef ', E2.Nom
FROM Employes E1,
     Employes E2 -- même table
WHERE COALESCE(E1.NumChef, E1.NumSecu)
           = E2.NumSecu
ORDER BY E2.Nom;
```

Recherche de base avec jointure : Auto-jointure

L'auto-jointure permet donc aussi d'implémenter l'intersection !

```sql
SELECT Ep1.NumSecu
FROM EmpPro Ep1, EmpPro Ep2
WHERE Ep1.NumSecu = Ep2.NumSecu
  AND Ep1.NumPro = 'p10346'
  AND Ep2.NumPro = 'p10349';


SELECT NumSecu FROM EmpPro WHERE NumPro = 'p10346'
INTERSECT
SELECT NumSecu FROM EmpPro WHERE NumPro = 'p10349';
```

Recherche de base avec jointure : Jointure naturelle

- La jointure s'effectue sur les colonnes communes, c'est-à-dire sur les colonnes qui portent le même nom [4].
- Le mot clé **USING** permet de restreindre les colonnes communes à prendre en considération
- Une jointure naturelle peut être une jointure **INNER**, **LEFT OUTER** ou **RIGHT OUTER**. La valeur par défaut est : **INNER**.

L'auto-jointure permet donc aussi d'implémenter l'intersection !

```
1 SELECT Colonnes
2 FROM table1
3         NATURAL JOIN table2 [USING col1, col2,...]
4 WHERE ...;
```

4. Même si l'information n'a rien à voir

Recherche de base avec jointure : Jointure naturelle

Afficher le nom des employés ainsi que le numéro des projets qui leur sont affectés.

```sql
SELECT Nom, NumPro
FROM Employes
        NATURAL JOIN EmpPro;
```

Recherche de base avec jointure : Jointure interne

Afficher le nom des employés ainsi que le numéro des projets auxquels ils sont affectés

```sql
SELECT Nom, NumPro
FROM Employes
        INNER JOIN EmpPro
                USING (NumSecu);


SELECT Nom, NumPro
FROM Employes
        INNER JOIN EmpPro
                ON (Employes.NumSecu = EmpPro.
                    NumSecu);
```

Recherche de base avec jointure : Jointure interne

Afficher le nom des employés ainsi que le NOM des projets auxquels ils sont affectés

```sql
SELECT Nom, NomPro AS NomProjet
FROM Employes
        INNER JOIN EmpPro USING (NumSecu)
        INNER JOIN Projets USING (NumPro);
```

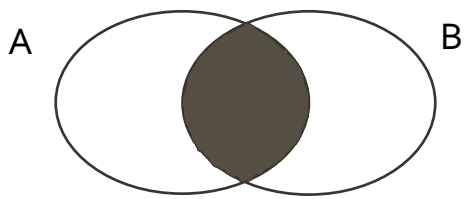

Figure 5 – intersection

Recherche de base avec jointure : Jointure externe

La jointure externe permet de récupérer les lignes des tables correspondant au critère de jointure, mais aussi celles pour lesquelles il n'existe pas de correspondance.

Jointure externe

```
SELECT colonnes
FROM TableGauche
  [RIGHT OUTER | LEFT OUTER | FULL OUTER] JOIN
  TableDroite ON condition
[WHERE prédicat]  ;
```

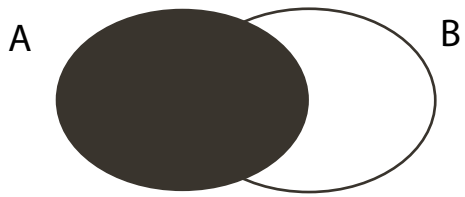

Figure 6 – Jointure externe gauche

Recherche de base avec jointure : Jointure externe

RIGHT OUTER : On cherche tous les tuples satisfaisant la condition de jointure précisée dans le prédicat, puis on rajoute toutes les lignes de la table TableDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

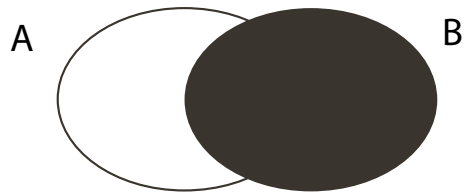

Figure 7 – Jointure externe droite

Recherche de base avec jointure : Jointure externe

LEFT OUTER : On cherche tous les tuples satisfaisant la condition de jointure précisée dans le prédicat, puis on rajoute toutes les lignes de la table TableGauche qui n'ont pas été prises en compte au titre de la satisfaction du critère.

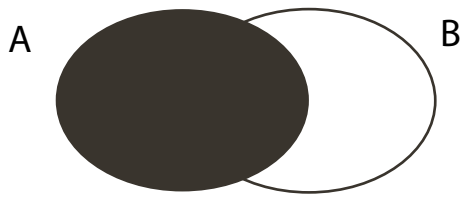

Figure 8 – Jointure externe gauche

Recherche de base avec jointure : Jointure externe

FULL OUTER : On cherche tous les tuples satisfaisant la condition de jointure précisée dans le prédicat, puis on rajoute toutes les lignes des tables TableGauche et TableDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

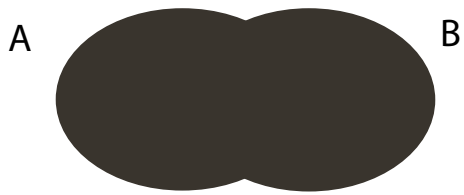

Figure 9 – Jointure externe gauche+droite

Recherche de base avec jointure : Jointure externe

Afficher le nom des employés ainsi que le numéro des projets auxquels ils sont affectés.

```
1 SELECT Employes.numsecu, nom, COALESCE(EmpPro.NumPro, '
      Pas de projet en cours') AS "Numéro de projet"
2 FROM Employes
3           LEFT OUTER JOIN EmpPro
4                       ON Employes.NumSecu = EmpPro.
                           NumSecu
5 ORDER BY 2;
```

5

5. Attention aux employés qui n'ont pas de projets en cours.

Recherche de base avec jointure : Jointure externe

Afficher le numéro et le nom des employés qui n'ont pas de projet en cours.

```sql
SELECT Employes.numsecu, nom
FROM Employes
        LEFT OUTER JOIN EmpPro
                        ON Employes.NumSecu = EmpPro.
                            NumSecu
WHERE EmpPro.NumSecu IS NULL;
```

Recherche de base avec jointure : Jointure externe

La jointure externe permet donc, entre autres, d'implémenter la différence :

> Afficher le numéro et le nom des employés qui n'ont pas de projet en cours.

```sql
SELECT Employes.numsecu, nom
FROM Employes
        LEFT OUTER JOIN EmpPro
                        ON Employes.NumSecu = EmpPro.
                            NumSecu
WHERE EmpPro.NumSecu IS NULL;

SELECT NumSecu
FROM Employes MINUS -- en Oracle 10g
SELECT NumSecu
FROM EmpPro;
```

Recherche de base avec jointure : Jointure d'union

Il n'y a pas de critère de jointure !

> La jointure d'union concatène les tables sans aucune correspondance de colonne

```sql
SELECT EmpPro.*, NULL, NULL, NULL
FROM EmpPro
UNION ALL
SELECT NULL, NULL, NULL, Departements.*
FROM Departements;
```

Recherche de base avec jointure : Remarque : clause FROM

Extension de SQL2 : possibilité de placer une liste de **`table-expression`** dans la clause **`FROM`**

Exemple : afficher le nom des projets développés par l'employé DE NIRO

```sql
SELECT NomPro
FROM (SELECT *
      FROM Employes
               NATURAL JOIN EmpPro
      WHERE UPPER(nom) = 'DE NIRO')
         JOIN Projets USING (NumPro);
```

Recherche de base avec jointure : Exercices

Exercices sur les jointures

- Rechercher le nom des employés et le nom du département dans lequel ils travaillent (le faire via une jointure manuelle, interne ; quid si on le fait avec une jointure naturelle ?)
- Rechercher les paires d'employés qui habitent la même commune
- Rechercher le nom des départements qui possèdent le même responsable
- Rechercher le nom des projets auxquels aucun employé n'est affecté
- Rechercher le nom des employés qui ne sont affectés à aucun projet

Recherche de base avec jointure : Exercices i

Rechercher le nom des employés et le nom du département dans lequel ils travaillent (le faire via une jointure manuelle et interne. Quid si on le fait avec une jointure naturelle ?)

```sql
-- Manuelle :
SELECT Nom, NomDep
FROM Employes,
     Departements
WHERE Employes.NumDep = Departements.NumDep;

-- Interne :
SELECT Nom, NomDep
FROM Employes
         INNER JOIN Departements
                    ON (Employes.NumDep = Departements.
                        NumDep);

```

Recherche de base avec jointure : Exercices ii

```sql
-- OU
SELECT Nom, NomDep
FROM Employes
        INNER JOIN Departements
                USING (NumDep);

-- Naturelle :
SELECT Nom, NomDep
FROM Employes
        NATURAL JOIN Departements;

-- Attention, dans ce cas-ci, on obtient moins de tuple
    car la jointure est faite sur le NumSecu en plus du
    NumDep, or la jointure sur le NumSecu ne doit pas
    intervenir dans ce cas-ci !!
```

Recherche de base avec jointure : Exercices

Rechercher les paires d'employés qui habitent la même commune

```sql
SELECT E1.Nom, E2.Nom, E1.Commune
FROM Employes E1,
     Employes E2
WHERE E1.Commune = E2.Commune
  AND E1.Nom < E2.Nom
ORDER BY E1.Nom;
```

Recherche de base avec jointure : Exercices

Rechercher le nom des départements qui possèdent le même responsable

```sql
SELECT D1.NomDep, D2.NomDep
FROM Departements D1
        INNER JOIN Departements D2 USING (NumSecu)
WHERE D1.NomDep < D2.NomDep;
```

Recherche de base avec jointure : Exercices

Rechercher le nom des projets auxquels aucun employé n'est affecté

```sql
SELECT NomPro
FROM Projets
        LEFT JOIN EmpPro USING (NumPro)
WHERE NumSecu IS NULL;
```

Recherche de base avec jointure : Exercices

Rechercher le nom des employés qui ne sont affectés à aucun projet

```sql
SELECT Nom
FROM Employes
        LEFT JOIN EmpPro USING (NumSecu)
WHERE NumPro IS NULL;
```

Expressions SQL

Table des matières de la section : Expressions SQL i

Table des matières de la section : Expressions SQL ii

Expressions SQL : Enrichir les clauses

- Enrichir les possibilités de la clause de sélection du **SELECT** :
  clause_de_selection ::= * | nom_table.* | liste_expression
- Enrichir les possibilités de la clause **WHERE**

Clause WHERE

```
condition_de_base ::=
 expression oper_comp expression

oper_comp ::=   = | <> | < | <= | > | >=

expression ::= expression_numérique
 | expression_caractère
 | expression_date_temps
 | expression_intervalle
```

Expressions SQL : Enrichir les clauses

Où ?

- Dans la clause ORDER BY
- Dans l'expression SELECT
- Dans l'instruction UPDATE … SET
- Dans l'instruction INSERT … VALUES
- Dans la clause WHERE

Expressions SQL : Expressions numériques

Expressions numériques

```
expression_numérique ::=
 terme | terme { + | - } terme

terme ::= facteur | terme {* | /} facteur

facteur ::= [+ | -] constante
  | [+ | -] nom_colonne
  | [+ | -] fonction_de_calcul
  | [+ | -] fonction_de_conversion
  | (expression_numérique)
```

Expressions SQL : Expressions numériques

Expressions numériques

```
fonction_de_calcul ::=
 COUNT()
 | AVG([ALL|DISTINCT] expression)
 | MAX([ALL|DISTINCT] expression)
 | MIN([ALL|DISTINCT] expression)
 | SUM([ALL|DISTINCT] expression)
 | COUNT([ALL|DISTINCT] expression)

fonction_de_conversion ::=
 CAST(expression AS type_de_donnée |
  domaine)
```

Expressions SQL : Expressions numériques i

Expressions numériques (fct de calcul) : `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`

- Dans la littérature anglo-saxone, ces fonctions sont appelées Aggregate functions.
- L'argument de `SUM` et AVG doit toujours être de type numérique.
- L'argument de `COUNT`, `MAX` et `MIN` peuvent être de type numérique, caractère ou date
- Toutes les valeurs indéfinies (`NULL`) de la collection sont toujours éliminées avant l'application de la fonctions À l'exception de `COUNT(*)` où toutes les valeurs indéfinies sont comptées.
- Si l'argument est la collection vide, `COUNT` donne 0, les autres fonctions donnent `NULL`

Expressions SQL : Expressions numériques

Expressions numériques

```sql
SELECT COUNT(NumSecu) AS NB,
       AVG(Bareme)    AS Moyenne,
       MIN(Bareme)    AS "Bareme Min",
       MAX(Bareme)    AS "Bareme Max"
FROM Employes
WHERE NumDep = 'd00004';
```

Expressions SQL : Expressions numériques

Afficher le nom de l'employé qui gagne le plus et le nom de l'employé qui gagne le moins - avec l'opérateur ensembliste UNION

```sql
SELECT Nom, Prenom, Bareme
FROM Employes
WHERE Bareme = (SELECT MIN(Bareme) FROM Employes)
UNION
SELECT Nom, Prenom, Bareme
FROM Employes
WHERE Bareme = (SELECT MAX(Bareme) FROM Employes);
```

Expressions SQL : Expressions numériques

Afficher le nom de l'employé qui gagne le plus et le nom de l'employé qui gagne le moins

```sql
SELECT E1.Nom, E1.Prenom, E1.Bareme
FROM Employes E1,
     (SELECT MIN(Bareme) AS BaremeMin FROM Employes) E2,
     (SELECT MAX(Bareme) AS BaremeMax FROM Employes) E3
WHERE E1.bareme = E2.BaremeMin
   OR E1.bareme = E3.BaremeMax;
```

Expressions SQL : Expressions caractères

Expressions caractères

```
expression_caractère :::
 'chaîne_constante'
| nom_colonne
| UPPER (expression_caractère)
| LOWER (expression_caractère)
| CHARACTER_LENGTH (expression_caractère)
| USER
| CAST (expression AS type_de_donnée|domaine)
| SUBSTRING (expression_caractère FROM début FOR long)
| expression_caractère {||expression_caractère }
| POSITION(expression_caractère IN expression_caractère)
| TRIM(ltb, [pad,] FROM expression_caractère)
```

Expressions SQL : Expressions caractères i

- `concat(ch1, ch2)` : cette fonction est identique à ||.
- `initcap(ch)` donne la chaîne ch dont le premier caractère a été converti en majuscules et les autres caractères en minuscules.
- `lower(ch)` : cette fonction est identique à celle de la norme.
- `lpad(ch1, x [, ch2])` construit une chaîne de x caractères en complétant le début de ch1 par le nombre adéquat de fois la chaîne ch2. Par défaut, ch2 est le caractère blanc.
- `ltrim(ch1 [, ch2])` retire du début de ch1 toutes les occurrences de ch2. Par défaut ch2, est le caractère blanc.
- `replace(ch1, ch2 [, ch3])` donne la chaîne ch1 dans laquelle toutes les occurrences de ch2 ont été remplacées par ch3

Expressions SQL : Expressions caractères ii

- `rpad(ch1, x [, ch2])` construit une chaîne de x caractères en complétant la fin de ch1 par le nombre adéquat de fois la chaîne ch2. Par défaut ch2, est le caractère blanc.
- `rtrim(ch1, ch2)` retire de la fin de ch1 toutes les occurrences de ch2. Par défaut ch2, est le caractère blanc.
- `substr(ch, x [, y])` extrait de ch, à partir de la position x, une sous-chaîne de longueur y. Si y est omis, substr extrait la sous-chaîne jusqu'à la fin de ch.
- `translate(ch1, ch2, ch3)` donne la chaîne ch1 dans laquelle toutes les occurrences de chaque caractère de ch2 ont été remplacées par le caractère correspondant de ch3.
- `upper(ch)` : cette fonction est identique à celle de la norme.

Expressions SQL : Expressions caractères iii

- `instr(ch1, ch2 [, x] [, y])` donne la position de ch2 dans ch1. La chaîne ch1 est parcourue à partir du xème caractère. Si y est précisé, instr donne la position de la yème occurrence de ch2 dans ch1. length(ch) est identique à `character_length`.

Expressions SQL : Opérateur CASE

Opérateur CASE

```
CASE expression
  WHEN valeur1 THEN resultat1
 WHEN valeur2 THEN resultat2
 ...
   [ ELSE resultatn ]
END
ou
CASE
  WHEN condition1 THEN resultat1
  WHEN condition2 THEN resultat2
  ...
  [ ELSE resultatn ]
END
```

Expressions SQL : Opérateur CASE

Rechercher le nom ainsi que la fourchette du salaire des employés dont le nom contient la lettre 'e' en deuxième position.

```sql
SELECT nom,
       (CASE
            WHEN bareme >= 90000 THEN '>= 90000'
            WHEN bareme >= 80000 THEN 'entre 80000 et
                89999'
            WHEN bareme >= 70000 THEN 'entre 70000 et
                79999'
            ELSE '< 70000'
           END) AS bareme,
       (CASE sexe
            WHEN 'F' THEN 'Féminin'
            ELSE 'Masculin'
           END) AS genre
FROM Employes
WHERE UPPER(nom) LIKE '_E%';
```

Expressions SQL : Expression de dates et temps

Expression de dates et temps

```
expression_date_temps ::=
        constante
 | nom_colonne
 | CAST ( expression AS type_de_donnée |  domaine)
 | CURRENT_DATE
 | CURRENT_TIME [ précision ]
 | CURRENT_TIMESTAMP [ précision ]
 | EXTRACT ( champ FROM source )
```

Expressions SQL : Expression de dates et temps

- `CURRENT_DATE` donne la date du jour selon le format 'aaaa-mm-jj'
- `CURRENT_TIME` donne l'heure courante selon le format 'hh:mm:ss'
- `CURRENT_TIMESTAMP` donne la date du jour et l'heure courante : CURRENT_DATE concaténée à `CURRENT_TIME`
- `EXTRACT ( champ FROM source )` Permet d'extraire la valeur numérique d'un champ d'une expression de type `date_temps` ou intervalle.
- Le paramètre champ peut valoir : `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`

Expressions SQL : Expression de dates et temps

Exemple : Rechercher le nom des employés nés dans les années 50

```sql
SELECT Nom, EXTRACT(YEAR FROM DateNais) "Année naissance
    "
FROM Employes
WHERE EXTRACT(YEAR FROM DateNais) BETWEEN 1950 AND 1959;
```

Expressions SQL : Expression de dates et temps

Lors d'une conversion de

- `DATE` en `TIMESTAMP`, la partie `TIME` est initialisée à `00:00:00.00`
  `TIME` en `TIMESTAMP`, la partie `DATE` est initialisée à `CURRENT_DATE`
  `TIMESTAMP` en `TIME`, on laisse tomber la partie `DATE`

Expressions SQL : Expression de dates et temps

- La fonction **CAST** permet de convertir un numérique vers une donnée de type datetime (**DATE**, **TIME**, **TIMESTAMP**) ou interval (année-mois ou jour-temps) :

  ```
  CAST (expression AS type_de_donnée | domaine)
  ```

- La fonction **EXTRACT** permet d'extraire l'année, le mois, le jour, les heures, minutes, secondes d'une donnée de type datetime ou interval

EXTRACT

```
EXTRACT ({YEAR | MONTH | DAY | HOUR |
      MINUTE | SECOND}
 FROM [datetime_value_expression |
  interval_value_expression})
```

Expressions SQL : Expression de dates et temps

Exemple CAST

```
CAST ('10/11/2017' AS DATE)
--Résultat : 10/11/2017 00:00:00

CAST ('10/11/2017' AS TIMESTAMP)
--Résultat : 10/11/2017 00:00:00.000000

CAST ('01 00:00:00' AS INTERVAL DAY TO SECOND)
--Résultat : +01 00:00:00.000000

CAST('01-11' As INTERVAL YEAR TO MONTH)
--Résultat : +01-11
```

Expressions SQL : Expression de dates et temps

Redéfinir le format des dates

```sql
ALTER SESSION
  SET NLS_DATE_FORMAT = 'DD/MM/YYYY HH24:MI:SS';
ALTER SESSION
  SET NLS_TIMESTAMP_FORMAT = 'DD/MM/YYYY HH24:MI:SS.FF';

SELECT TO_DATE ('10/11/2017', 'DD/MM/YYYY') FROM DUAL;
--Résultat : 10/11/2017 00:00:00

SELECT TO_TIMESTAMP ('10/11/2017', 'DD/MM/YYYY') FROM
    DUAL;
--Résultat : 10/11/2017 00:00:00.000000
```

Expressions SQL : Temps et environnements distribués

- SQL2 donne un ensemble de spécifications permettant de gérer correctement les problèmes de temps dans un environnement distribué.
- L'idée de base est que chaque utilisateur mémorise le temps GMT (Greenwich Mean Time) mais qu'il peut le visualiser et l'utiliser comme son temps local

Expressions SQL : Temps et environnements distribués

Voici par exemple comment deux utilisateurs situés respectivement à San Francisco et à Helsinki peuvent manipuler la table T définie de la manière suivante :

Redéfinir le format des dates

```
1 CREATE TABLE T
2 ( ...
3   debut TIME WITH TIME ZONE,
4   ...);
```

Expressions SQL : Temps et environnements distribués

A San Francisco	représentation interne	A Helsinki
SET TIME ZONE -8:00		**SET TIME ZONE** +2:00
! Il est 10 heures		
INSERT INTO T (..., debut, ...) **VALUES** (..., **TIME** '10:00:00', ...);		
	TIME '18:00:00'	
		SELECT debut **FROM** T;
		TIME '20:00:00'

Figure 10 – Temps et environnements distribués

Expressions SQL : Expressions d'intervalles

1er opérande	Opérateur	2ème opérande	Résultat
Datetime	-	Datetime	Intervalle
Datetime	+	Intervalle	Datetime
Datetime	-	Intervalle	Datetime
Intervalle	+	Datetime	Datetime
Intervalle	+	Intervalle	Intervalle
Intervalle	-	Intervalle	Intervalle
Intervalle	*	Numérique	Intervalle
Intervalle	/	Numérique	Intervalle
Numérique	*	Intervalle	intervalle

Figure 11 – Opérations autorisées

Expressions SQL : Expressions d'intervalles i

Exemple : afficher l'âge des employés

```sql
SELECT nom,
       DateNais,
       EXTRACT(YEAR FROM CURRENT_DATE)
           - EXTRACT(YEAR FROM DateNais) Age
FROM Employes;

SELECT nom,
       DateNais,
       (CURRENT_DATE - DateNais) YEAR TO MONTH Age
FROM Employes;


SELECT nom,
       DateNais,
```

Expressions SQL : Expressions d'intervalles ii

```
15          EXTRACT(YEAR FROM (CURRENT_DATE - DateNais)
16              YEAR TO MONTH)
17              || ' years ' ||
18          EXTRACT(MONTH FROM (CURRENT_DATE - DateNais)
19              YEAR TO MONTH)
20              || ' months ' AS Age
21  FROM Employes;
```

7

7. ATTENTION : le nombre d'années entre le 31/12/2018 et le 01/01/2019 vaut 1 !

Expressions SQL : L'opérateur OVERLAPS i

- SQL2 possède un opérateur spécial : OVERLAPS.
- Overlaps permet de tester si deux périodes de temps se recouvrent. Ces périodes de temps peuvent être représentées de deux manières différentes : un temps de départ et un temps de fin ou un temps de départ et un intervalle.
- Pour plus d'info, voir les documents de référence.

Tri

Table des matières de la section : Tri

Table des matières de la section : Tri ii

Tri : Syntaxe

Tri

```
instruction_de_sélection ::=
          expression_de_sélection [ clause_de_tri ] ;

clause_de_tri ::= ORDER BY liste_colonne_tri

colonne_tri ::= colonne | numéro [ ASC | DESC ]
```

Tri : Syntaxe

- Lors d'un tri, toutes les valeurs indéfinies (NULL) sont considérées comme étant identiques.
- Lors d'un tri, la valeur indéfinie (NULL) est considérée comme inférieure ou supérieur à toute valeur définie. Ce choix est laissé au constructeur.

En Oracle, NULL est supérieure [8] à toute valeur définie.

8. On peut aussi préciser l'emplacement des NULL dans le résultat du tri en précisant NULL FIRST | NULL LAST

Tri : Syntaxe

Exemple : Afficher et trié par âge décroissant le nom des employés du départements 'Application telecom'

```
1 SELECT Nom, DateNais,
2   (CURRENT_DATE - DateNais) YEAR TO MONTH AS Age
3 FROM Employes, Departements
4 WHERE Employes.NumDep = Departements.NumDep
5   AND NomDep = 'Applications telecom'
6 ORDER BY Age DESC;
7
8 --Ou ORDER BY 3 DESC
```

Tri : Syntaxe

Afficher et triés par commune le nom des employés habitant à Ans Neupré ou Chaudfontaine. Les communes doivent apparaître dans l'ordre Neupré Ans puis Chaudfontaine.

```
1 SELECT Nom, Commune
2 FROM Employes
3 WHERE Commune IN
4       ('NEUPRE', 'ANS', 'CHAUDFONTAINE')
5 ORDER BY (CASE Commune
6                 WHEN 'NEUPRE' THEN '0'
7                 WHEN 'ANS' THEN '1'
8                 ELSE '2'
9     END), Nom;
```

Groupement de lignes

Table des matières de la section : Groupement de lignes i

Table des matières de la section : Groupement de lignes ii

Groupement de lignes : Syntaxe

Groupement de lignes

```
expression_de_sélection ::=
    SELECT [ ALL | DISTINCT ] clause_de_sélection
    FROM liste_table_ref
    [ WHERE condition ]
    [ clause_grouper ]
    [ clause_sélection_groupes ];

clause_grouper ::= GROUP BY liste_colonne

clause_sélection_groupes ::= HAVING condition
```

Groupement de lignes : Syntaxe

- Appelons T le résultat de l'évaluation des clauses `FROM`, `WHERE`.
- Chaque colonne mentionnée dans la clause `GROUP BY` doit être une colonne de T.

Groupement de lignes : Syntaxe

- Le résultat de la clause `GROUP BY` est une table intermédiaire constituée d'un ensemble de groupes de lignes issus de T de la manière suivante.
- Les lignes de *T* sont (conceptuellement) réarrangées en groupes de telle sorte que dans chaque groupe toutes les lignes possèdent la même valeur pour la combinaison de colonnes indiquées dans la clause `GROUP BY`.
- Chaque item de la clause `SELECT` doit fournir une seule valeur par groupe : `COUNT`, `SUM`, `AVG`, `MAX` ou `MIN`

Groupement de lignes : Syntaxe

- `HAVING` permet d'indiquer une condition qui sera évaluée pour chaque groupe formé en fonction de la clause `GROUP BY`. Les groupes qui ne satisfont pas la condition sont éliminés.
- Les colonnes présentes dans la condition de la clause `HAVING` doivent être présentes dans la clause `GROUP BY`, ou argument d'une fonction, ou être une référence externe .

Groupement de lignes : L'ordre d'évaluation

1. `FROM`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `SELECT`
6. `ORDER BY`

Groupement de lignes : Exemples

Exemple : Afficher le nombre d'employés par département

```sql
SELECT NumDep, COUNT(*) NbEmp
FROM Employes
GROUP BY NumDep;

--Mieux :
SELECT COALESCE(NumDep, 'Inconnu') NrDep,
       COUNT(*)                    NbEmp
FROM Employes
GROUP BY NumDep;
```

Groupement de lignes : Exemples

Rechercher le nom le prénom et le nombre total d'heures hebdomadaires pour chaque employé et ce trié par nombre d'heures

```sql
SELECT nom, prenom, SUM(Heures) NbHeures
FROM Employes JOIN EmpPro USING (NumSecu)
GROUP BY NumSecu;
-- Ceci Provoque une erreur
-- Tous les champs qui apparaissent dans le SELECT
    doivent apparaître dans le GROUP BY

SELECT nom, prenom, SUM(Heures) NbHeures
FROM Employes JOIN EmpPro USING (NumSecu)
GROUP BY NumSecu, nom, prenom
ORDER BY 3;
```

Groupement de lignes : Exemples

Exemple : Rechercher le nom le prénom et le nombre total d'heures hebdomadaires pour les employés qui ont une charge < à 40 heures

```sql
SELECT nom, prenom, SUM(Heures) NbHeures
FROM Employes
        JOIN EmpPro USING (NumSecu)
GROUP BY NumSecu, nom, prenom
HAVING SUM(Heures) < 40
ORDER BY 3;
-- On ne peut pas utiliser l'alias de la colonne dans le
    HAVING : le SELECT n'a pas encore été exécuté, donc
   cet alias n'est pas encore connu.
```

Groupement de lignes : Exemples

Exemple : Rechercher le minimum d'heures hebdomadaires

```sql
SELECT MIN(SUM(heures)) NbHeures
FROM EmpPro
GROUP BY NumSecu;
```

Groupement de lignes : Exemples

Exemple : Rechercher le nom et le prénom des employés qui ont la plus petite charge hebdomadaire

```sql
SELECT Nom, Prenom, MIN(SUM(heures)) NbHeures
FROM Employes JOIN EmpPro USING (NumSecu)
GROUP BY NumSecu, Nom, Prenom;
--Provoque une erreur - ORA-00937
```

Provoque une erreur à cause du `MIN(SUM(heures))` Voir la solution au slide 108.

Groupement de lignes : Exemples

Exemple : Rechercher le nom et le prénom des employés qui ont la plus petite charge hebdomadaire

```sql
CREATE
OR REPLACE VIEW Temp AS (
  SELECT Nom, Prenom, SUM(Heures) NbHeures
  FROM Employes JOIN EmpPro USING (NumSecu)
  GROUP BY NumSecu, Nom, Prenom);

SELECT *
FROM Temp
WHERE NbHeures = (SELECT MIN(NbHeures)
                  FROM Temp);
```

9

9. Pensez à ajouter les privilèges suffisants.`GRANT CREATE ANY VIEW TO xx;`

Groupement de lignes : Exemples

Autre solution sans utiliser les vues :

Exemple : Rechercher le nom et le prénom des employés qui ont la plus petite charge hebdomadaire

```
1 SELECT Nom, Prenom, SUM(Heures) NbHeures
2 FROM Employes
3           JOIN EmpPro USING (NumSecu)
4 GROUP BY NumSecu, Nom, Prenom
5 HAVING SUM(Heures) =
6        (SELECT MIN(SUM(Heures))
7         FROM Employes
8                   JOIN EmpPro USING (NumSecu)
9         GROUP BY NumSecu);
```

Groupement de lignes : Exemples

Autre solution sans utiliser les vues :

Exemple : Afficher le maximum de la moyenne des salaires par département

```sql
SELECT MAX (AVG(Bareme))
FROM Employes;
--Provoque une erreur !!!

SELECT MAX(AVG(Bareme)) AS MaxAvg
FROM Employes
GROUP BY NumDep;
--OU
SELECT MAX(Temp.Moyenne)
FROM (SELECT AVG(Bareme) AS Moyenne
      FROM Employes
      GROUP BY NumDep) Temp;
```

Groupement de lignes : Exemples

Autre solution sans utiliser les vues :

Exemple : Afficher le numéro de département qui a la moyenne des barèmes la plus élevée

```sql
SELECT NumDep, MAX(AVG(Bareme)) AS MaxAvg
FROM Employes
GROUP BY NumDep;
-- Provoque une erreur ! MAX(AVG(Bareme)) - ORA-00937

SELECT NumDep, AVG(Bareme) AS MaxAvg
FROM Employes
GROUP BY NumDep
HAVING AVG(Bareme) = (SELECT MAX(AVG(Bareme))
                      FROM Employes
                      GROUP BY NumDep);
```

Sélections imbriquées

Sélections imbriquées : Pourquoi

- Les sélections imbriquées représentent un concept du SQL qui contribue grandement à sa puissance et à sa souplesse : la sélection imbriquée ou sous-question.
- Une sélection imbriquée n'est rien d'autre qu'un bloc de qualification SFW encapsulé à l'intérieur d'un autre bloc de qualification.

Sélections imbriquées : Syntaxe

Sélections imbriquées

```
instruction_de_sélection ::=
        expression_de_sélection  [ clause_de_tri ];
expression_de_sélection ::=
      SELECT clause_de_sélection FROM liste_table_ref
      [ WHERE condition ]
      [ clause_grouper ]
      [ clause_sélection_groupes ]

condition ::=
      [ NOT ] condition_élémentaire
     | condition AND | OR condition
     | ( condition )
condition_élémentaire ::=
        condition_de_base | condition_between |
            condition_in | condition_like |
            condition_null | condition_all_any |
            condition_exists
```

Sélections imbriquées : Syntaxe

```
condition_in ::= expression [ NOT ] IN (sélection_une_colonne)
```

Le résultat de l'évaluation de la condition `condition_in` est **VRAI** si et seulement si la valeur de l'expression à gauche du `IN` est égale à au moins une des valeurs du résultat du `sélection_une_colonne`.

Sélections imbriquées : Exemples

Exemple : Rechercher le nom et le prénom des employés qui travaillent sur le projet p10347

```sql
SELECT Nom, Prenom
FROM Employes
WHERE NumSecu IN (SELECT NumSecu
                  FROM EmpPro
                  WHERE NumPro = 'p10347');
```

Sélections imbriquées : Exemples

Exemple : Afficher le numéro des employés qui travaillent sur les projets 'p10346' et 'p10349'

```
1 SELECT NumSecu
2 FROM EmpPro
3 WHERE NumPro = 'p10346'
4   AND NumSecu IN (SELECT NumSecu
5                   FROM EmpPro
6                   WHERE NumPro = 'p10349');
```

Sélections imbriquées : Exemples

Exemple : Afficher le numéro des employés qui n'ont pas de projets en cours

```sql
SELECT NumSecu
FROM Employes
WHERE NumSecu NOT IN (SELECT NumSecu
                      FROM EmpPro);
```

Sélections imbriquées : Exemples

Exemple : Rechercher le nom des départements qui n'ont pas d'employé qui gagne moins de 60000

```
1 SELECT NomDep
2 FROM Departements
3 WHERE NumDep NOT IN (SELECT NumDep
4                      FROM Employes
5                      WHERE Bareme < 60000);
```

Sélections imbriquées : Exemples

Important

- Dans `Employes` il existe des employés sans numéro de département et donc la requête imbriquée donne un numéro de département et `NULL`
- Quand on fait la comparaison avec `NULL` dans le `NOT IN`, on obtient des résultats incorrects !

Sélections imbriquées : Exemples

Exemple : Rechercher le nom des départements qui n'ont pas d'employé qui gagne moins de 60000

```sql
SELECT NomDep
FROM Departements
WHERE NumDep NOT IN
      (SELECT COALESCE(NumDep, 'Sans Numéro')
       FROM Employes
       WHERE Bareme < 60000);
```

Sélections imbriquées : Exemples

Exemple : Rechercher le nom et le prénom des employés qui travaillent 10 heures sur le projet p10349

```sql
SELECT Nom, Prenom
FROM Employes
WHERE (NumSecu, 10, 'p10349') IN
      (SELECT NumSecu, Heures, NumPro
       FROM EmpPro);
```

Sélections imbriquées : Remarques

`condition_all_any ::=expression oper_comp [ ALL | ANY | SOME ] ( sélection_une_colonne )` Le résultat de l'évaluation de `oper_comp ALL` est **VRAI** uniquement dans un des deux cas suivants :

- La comparaison indiquée par `oper_comp` est VRAIE pour toutes les valeurs du résultats du `selection_une_colonne`
- Le résultat du `selection_une_colonne` est vide

Sélections imbriquées : Remarques

`condition_all_any ::= expression oper_comp [ ALL | ANY | SOME ] ( sélection_une_colonne )` Le résultat de l'évaluation de `oper_comp ANY (SOME)` est **VRAI** si et seulement si la comparaison indiquée par `oper_comp` est VRAIE pour au moins une des valeurs du résultat du `selection_une_colonne`.

Sélections imbriquées : Remarques

```
condition_all_any ::= expression oper_comp [ ALL | ANY | SOME
] ( sélection_une_colonne )
```

- Le résultat de l'évaluation de `oper_comp ANY (SOME)` est **FAUX** si la comparaison indiquée par `oper_comp` est FAUSSE pour chaque valeur du résultat du `selection_une_colonne`.
- Dans tous les autres cas, le résultat de `oper_comp ANY (SOME)` est INCONNU

Sélections imbriquées : Exemples

Exemple : Afficher le numéro des employés qui travaillent sur les projets p10346 et p10349

```
1 SELECT NumSecu
2 FROM EmpPro
3 WHERE NumPro = 'p10346'
4   AND NumSecu =
5     ANY (SELECT NumSecu
6          FROM EmpPro
7          WHERE NumPro = 'p10349');
```

Sélections imbriquées : Exemples

Exemple : Afficher le numéro des employés qui n'ont pas le plus petit salaire

```
1 SELECT NumSecu
2 FROM Employes
3 WHERE bareme > ANY (SELECT bareme
4                     FROM Employes);
```

Sélections imbriquées : Exemples

Exemple : Afficher le numéro des employés qui n'ont pas de projet en cours

```
1 SELECT NumSecu
2 FROM Employes
3 WHERE NumSecu <> ALL (SELECT NumSecu
4                       FROM EmpPro);
```

Sélections imbriquées : Exemples

Exemple : Afficher le numéro des employés qui gagnent plus que tous les employés du département d00002

```sql
SELECT NumSecu
FROM Employes
WHERE Bareme > ALL (SELECT Bareme
                    FROM Employes
                    WHERE NumDep = 'd00002');
```

Sélections imbriquées : Exemples

Exemple : Afficher le nom des employés qui ont le salaire le plus élevé

```sql
SELECT Nom
FROM Employes
WHERE Bareme >=
          ALL (SELECT Bareme
               FROM Employes
               WHERE bareme IS NOT NULL);
```

Sélections imbriquées : Exemples

Exemple : Afficher le nom et le prénom des employés qui ont la plus petite charge hebdomadaire

```sql
CREATE
OR REPLACE VIEW Temp AS (
SELECT Nom, Prenom, SUM(heures) NbHeures
FROM Employes JOIN EmpPro USING (NumSecu)
GROUP BY NumSecu, Nom, Prenom);

SELECT *
FROM Temp
WHERE NbHeures = (SELECT MIN(NbHeures)
                  FROM Temp);
```

Sélections imbriquées : Exemples

Exemple : Afficher le nom et le prénom des employés qui ont la plus petite charge hebdomadaire

```sql
SELECT Nom, Prenom, SUM(Heures) NbHeures
FROM Employes
         JOIN EmpPro USING (NumSecu)
GROUP BY NumSecu, Nom, Prenom
HAVING SUM(Heures) <=
           ALL (SELECT SUM(Heures)
                FROM EmpPro
                GROUP BY NumSecu)
ORDER BY 3;
```

Sélections imbriquées : Sous-requêtes corrélées

Important

- Il existe des requêtes pour lesquelles la sous question référence le niveau externe.
- Dans ce type de requêtes, il est impossible d'obtenir le résultat de la sous-question sans faire référence au niveau externe.
- On parle de **sous questions corrélatives**.

Sélections imbriquées : Exemples

Exemple : Afficher le nom et le prénom des employés qui ont travaillé 10 heures sur le projet p10349

```
1 SELECT Nom, Prenom
2 FROM Employes
3 WHERE ('10', 'p10349') IN
4       (SELECT Heures, NumPro
5        FROM EmpPro
6        WHERE EmpPro.NumSecu =
7              Employes.NumSecu);
```

Sélections imbriquées : Exemples

Important

1. Le SGBD examine la première ligne de la table Employes.
2. La variable `Employes.NumSecu` vaut : 451278
3. Le système évalue le bloc interne : `(SELECT Heures, NumPro FROM EmpPro WHERE EmpPro.NumSecu = 451278);`
4. $\{(10, p10345), (12, p10346), \mathbf{(10, p10349)}, (8, p10351)\}$
5. L'employé 451278, Célarié Clémentine fait donc partie de la réponse.
6. Le système répète le traitement pour toutes les lignes de la table employés.

Sélections imbriquées : Exemples

Exemple : Rechercher le nom des employés qui ne sont pas attachés au même département que leur chef

```sql
SELECT Nom
FROM Employes E1
WHERE NumDep <> (SELECT NumDep
                 FROM Employes
                 WHERE NumSecu = E1.NumChef);

SELECT Nom
FROM Employes E1
WHERE COALESCE(NumDep, 'X') <>
              (SELECT COALESCE (NumDep, 'X')
               FROM Employes
               WHERE NumSecu = E1.NumChef);
```

Sélections imbriquées : Exemples

Exemple : Rechercher le nom et le prénom des employés dont le salaire est supérieur au salaire moyen de leur département

```sql
SELECT Nom, Prenom
FROM Employes E1
WHERE Bareme >
      (SELECT AVG(bareme)
       FROM Employes E2
       WHERE E1.NumDep = E2.NumDep);
```

Sélections imbriquées : Exemples

Exemple : Rechercher le nom et le prénom des employés qui ont travaillé 10 heures sur le projet p10349

```sql
SELECT Nom, Prenom
FROM Employes
WHERE 0 <
      (SELECT COUNT(*)
       FROM EmpPro
       WHERE heures = 10
         AND NumPro = 'p10349'
         AND EmpPro.NumSecu = Employes.NumSecu);
-- ou avec une jointure
select nom, PRENOM
from EMPLOYES
         inner join EMPPRO E on EMPLOYES.NUMSECU = E.
             NUMSECU
where HEURES = 10
  AND NumPro = 'p10349';
```

Sélections imbriquées : Exemples

Exemple : Rechercher le numéro des employés qui ont travaillé sur les projets p10346 et p10349

```
1 SELECT NumSecu
2 FROM EmpPro EP1
3 WHERE NumPro = 'p10346'
4   AND 1 = (SELECT COUNT (*)
5            FROM EmpPro
6            WHERE NumPro = 'p10349'
7            AND EmpPro.NumSecu = EP1.NumSecu);
```

Sélections imbriquées : Exemples

Exemple : Rechercher le numéro des employés qui n'ont pas de projet en cours

```
1 SELECT NumSecu
2 FROM Employes
3 WHERE 0 =
4       (SELECT COUNT(*)
5        FROM EmpPro
6        WHERE EmpPro.NumSecu = Employes.NumSecu);
```

Sélections imbriquées : Exemples

Exemple : Rechercher le nom et le prénom des employés qui ont la plus petite charge hebdomadaire

```sql
SELECT Nom, Prenom
FROM Employes
WHERE (SELECT SUM(Heures)
       FROM EmpPro
       WHERE EmpPro.NumSecu = Employes.NumSecu)
          =
      (SELECT MIN(SUM(Heures))
       FROM EmpPro
       GROUP BY NumSecu);
```

Utilisation de ”EXISTS”

Table des matières de la section : Utilisation de ”EXISTS” i

Table des matières de la section : Utilisation de ”EXISTS” ii

Utilisation de ”EXISTS” : Tester l’existence de données

Tester l’existence de données

- `condition_exists ::= EXISTS (selection_une_colonne)`
- La valeur de `condition_exists` est **VRAIE** si et seulement si l’évaluation de `sélection_une_colonne` n’est pas vide. Sinon, la valeur de `condition_exists` est **FAUSSE**
- L’opérateur `EXISTS` est de loin l’opérateur le plus puissant de SQL
- On peut exprimer des jointures
- On peut réaliser des intersections, différences, divisions

Utilisation de ”EXISTS” : Exemples

Exemple : Rechercher le nom et le prénom des employés qui ont travaillé 10 heures sur le projet p10349

```sql
SELECT Nom, Prenom
FROM Employes
WHERE EXISTS
  (SELECT * -- Constante arbitraire
   FROM EmpPro
   WHERE NumSecu = Employes.NumSecu -- Ici il y a de la
       corrélation
     AND Heures = 10
     AND NumPro = 'p10349');
```

Utilisation de ”EXISTS” : Exemples

Exemple : Rechercher le numéro des employés qui ont travaillé sur les projets p10346 et p10349

```
1 SELECT NumSecu
2 FROM EmpPro EP1
3 WHERE NumPro = 'p10346'
4   AND EXISTS(SELECT *
5              FROM EmpPro
6              WHERE NumPro = 'p10349'
7                AND EmpPro.NumSecu =
8                    EP1.NumSecu);
```

Utilisation de ”EXISTS” : Exemples

Exemple : Rechercher le numéro des employés qui ne sont attachés à aucun projet

```
1 SELECT NumSecu,NOM,PRENOM
2 FROM Employes
3 WHERE NOT EXISTS
4            (SELECT *
5             FROM EmpPro
6             WHERE EmpPro.NumSecu =
7                       Employes.NumSecu);
```

Utilisation de ”EXISTS” : Exemples

Exemple : Rechercher le nom des départements qui ont au moins un employé qui gagne plus de 90000

```
1 SELECT NomDep
2 FROM Departements
3 WHERE EXISTS
4           (SELECT *
5            FROM Employes
6            WHERE NumDep = Departements.NumDep
7              AND Bareme > 90000);
```

10

10. Images Numeriques est un département qui n’a pas d’employés => ne devrait pas apparaître dans le résultat !

Utilisation de ”EXISTS” : Exemples

Exemple : Rechercher le nom des départements qui n’ont aucun employé qui gagne plus de 90000

```
1  SELECT NomDep
2  FROM Departements
3  WHERE NOT EXISTS
4      (SELECT *
5       FROM Employes
6       WHERE NumDep = Departements.NumDep -- Ici il y a de
              la corrélation
7         AND Bareme > 90000)
8    AND EXISTS
9      (SELECT *
10      FROM Employes
11      WHERE NumDep = Departements.NumDep);-- Ici il y a
            de la corrélation
```

Utilisation de ”EXISTS” : Exemples

Exemple : Afficher le nom des employés qui ont le salaire le plus élevé

```
1 SELECT Nom, PRENOM
2 FROM Employes E1
3 WHERE NOT EXISTS
4     (SELECT *
5      FROM Employes
6      WHERE Bareme > E1.bareme);
```

11

11. L’employé MONROE a un salaire inconnu $\Longrightarrow$ ne devrait pas apparaître dans le résultat. Mais avec NOT EXISTS, la condition est vraie si le résultat de la sélection est vide !

Utilisation de ”EXISTS” : Exemples

Exemple : Afficher le nom des employés qui ont le salaire le plus élevé

```sql
SELECT Nom
FROM Employes
WHERE bareme = (SELECT MAX(bareme)
                FROM Employes);
```

Utilisation de ”EXISTS” : Exemples

Exercices récapitulatifs :

- Afficher le numéro des employés qui travaillent sur les projets p10346 et p10349
- écrire de 6 manières différentes :
  - opérateur ensembliste ;
  - auto-jointure ;
  - requête imbriquée ;
  - ALL/ANY/SOME ;
  - EXISTS ;
  - requête corrélée sans EXISTS.

Utilisation de ”EXISTS” : Exemples

Exercices récapitulatifs :

- Afficher le numéro des employés qui n’ont pas de projet en cours
- écrire de 6 manières différentes :
  - opérateur ensembliste ;
  - auto-jointure ;
  - requête imbriquée ;
  - ALL/ANY/SOME ;
  - EXISTS ;
  - requête corrélée sans EXISTS.

Mise à jour des données

Table des matières de la section : Mise à jour des données i

Table des matières de la section : Mise à jour des données ii

Mise à jour des données : INSERT

INSERT

```
instruction_d_ajout ::=

INSERT INTO nom_table [ ( liste_colonne ) ]
  VALUES ( liste_valeur )
        | expression_de_sélection ;
```

Mise à jour des données : INSERT

On souhaite ajouter un nouveau département - Il s'agit du département "Infographie" dont on ne connaît pas le chef.

```
1 INSERT INTO departements (numdep,nomdep,numsecu)
2      VALUES ('d00006', 'Infographie', NULL);
3
4 INSERT INTO departements (numdep,nomdep)
5      VALUES ('d00006', 'Infographie');
```

12

12. Remarque : les colonnes pour lesquelles on ne spécifie pas explicitement une valeur sont initialisées à leur valeur par défaut.

Mise à jour des données : INSERT

Exemple : Insertion d'un employé

```
1 INSERT INTO Employés
2 (NumSecu, Nom, Prenom, DateNais, Sexe,
3  Adresse, CodePostal, Commune, Bareme)
4 VALUES ('1234567', 'HOFFMAN', 'Dustin',
5         TO_DATE('1949-03-19', 'YYYY-MM-DD'), 'M',
6         'rue Lantin, 163', '4430', 'ANS', 60000);
```

Mise à jour des données : INSERT

Insérer l'employé DUBOIS, Luc. Ce nouvel employé est attaché au département où il y a le moins d'employés, il a le barème moyen des employés et a pour chef son ami HOFFMAN

Mise à jour des données : INSERT

Exemple : Insérer l'employé DUBOIS Luc

```
INSERT INTO employes
    (numsecu, nom, prenom, numdep, bareme, numchef)
VALUES ('888888', 'DUBOIS', 'Luc',
        (SELECT numdep
         FROM employes
         WHERE numdep IS NOT NULL
         GROUP BY numdep
         HAVING COUNT(*) <= ALL
                (SELECT COUNT(*)
                 FROM employes
                 WHERE numdep IS NOT NULL
                 GROUP BY numdep)),
        (SELECT AVG(bareme) FROM employes),
        (SELECT numsecu FROM employes WHERE nom = '
            HOFFMAN'));
```

Mise à jour des données : UPDATE

UPDATE

```
instruction_de_modification :::=
UPDATE nom_table
SET liste_colonne_valeur
[ WHERE condition ] ;

liste_colonne_valeur :::=
  nom_colonne = expression
```

Mise à jour des données : UPDATE

Exemple : Augmenter de 10% le barèmes des employés du département 'Applications telecom'

```
1 UPDATE employes
2 SET bareme = bareme * 1.10
3 WHERE numdep =
4       (SELECT numdep
5        FROM departements
6        WHERE nomdep = 'Applications telecom');
```

Mise à jour des données : UPDATE

Exemple : Attribuer à DE NIRO le salaire maximum du département auquel il est attaché

```
1  UPDATE employes emp1
2  SET bareme = (SELECT MAX(bareme)
3                FROM employes
4                WHERE numdep = emp1.numdep)
5  WHERE nom = 'DE NIRO'
6    AND prenom = 'Robert';
7
8  SELECT numsecu, bareme
9  FROM employes
10 WHERE nom = 'DE NIRO'
11   AND prenom = 'Robert';
```

Mise à jour des données : DELETE

DELETE

```
instruction_de_suppression :::=

DELETE FROM nom_table [ WHERE condition ] ;
```

Mise à jour des données : DELETE

Exemple : Supprimer toutes les attributions de Clémentine CELARIE à un projet

```sql
DELETE
FROM EmpPro
WHERE numsecu
          IN (SELECT numsecu
              FROM employes
              WHERE nom = 'CELARIE'
                AND prenom = 'Clémentine');
```