

# **SGBD - 2<sup>e</sup>**

## Chapitre 4 - Langage de Manipulation des Données (LMD)

---

Anne Léonard

6 septembre 2023

Haute École de la Province de Liège

# Table des matières du chapitre i

1. Introduction
2. Recherche de base
3. Recherche de base avec jointure
4. Expressions SQL
5. Tri
6. Groupement de lignes
7. Sélections imbriquées

## Table des matières du chapitre ii

8. Utilisation de "EXISTS"

9. Mise à jour des données

# Introduction

---

# Table des matières de la section : Introduction i

## 1. Introduction

### 1.1 Définition

### 1.2 Remarques

## 2. Recherche de base

## 3. Recherche de base avec jointure

## 4. Expressions SQL

## 5. Tri

## 6. Groupement de lignes

## Table des matières de la section : Introduction ii

7. Sélections imbriquées

8. Utilisation de "EXISTS"

9. Mise à jour des données

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.

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<sup>1</sup>.

- 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**

---

## 1. Introduction

## 2. Recherche de base

### 2.1 SELECT

### 2.2 WHERE

### 2.3 LIKE

### 2.4 NULL

### 2.5 Union

### 2.6 EXCEPT

### 2.7 INTERSECT

### 2.8 Exercices

## Table des matières de la section : Recherche de base ii

3. Recherche de base avec jointure

4. Expressions SQL

5. Tri

6. Groupement de lignes

7. Sélections imbriquées

8. Utilisation de "EXISTS"

9. Mise à jour des données

# Recherche de base : SELECT

## Clause SELECT

```
1 select_de_base ::=  
2     SELECT [ ALL | DISTINCT ] clause_de_sélection  
3     FROM nom_table  
4     [ WHERE condition ]  
5     ;  
6  
7     clause_de_sélection ::=  
8     * | nom_table.* | liste_colonne
```

# Recherche de base : WHERE

La clause WHERE permet de spécifier un critère de sélection.

Clause WHERE

```
1 condition ::=  
2   [ NOT ] condition_de_base  
3   | condition_between  
4   | condition_in  
5   | condition_like  
6   | condition_null  
7   | condition AND | OR condition  
8   | ( condition )
```

# Recherche de base : WHERE

Les possibilités de la clause WHERE sont étendues en développant la syntaxe de **condition\_de\_base**.

Condition de base

```
1 condition_de_base ::=  
2     expression oper_comp expression  
3  
4 oper_comp ::= = | <> | < | <= | > | >=  
5  
6 expression ::= expression_numérique  
7     | expression_caractère  
8     | expression_date_temps  
9     | 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 comparaison expression.

comparaison expression

```
1 row_constructor comparison_operator row_constructor
2
3 comparison_operator ::=      = | <> | < | <= | > | >=
4
5 row_constructor ::= liste de valeurs
6 | ( 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

```
1 SELECT Nom
2 FROM Employes
3 WHERE bareme >
4     (SELECT bareme
5      FROM Employes
6      WHERE nom = 'BEART');
7
8 SELECT Nom
9 FROM Employes
10 WHERE (sexe, bareme, numdep) =
11     (SELECT DISTINCT 'F', 80000, 'd000001'
12      FROM Employes);
```

# Recherche de base : LIKE

L'opérateur LIKE permet de comparer des chaînes de caractères

```
1 nom_col [NOT] LIKE
2   'modèle de chaîne'
3   [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

```
1 CREATE TABLE chaines
2 (
3     NomChar4      CHAR(4),
4     NomChar10     CHAR(10),
5     NomVarChar4   VARCHAR(4),
6     NomVarChar10  VARCHAR(10)
7 );
8
9 INSERT INTO chaines
10 VALUES ('SQL', 'SQL', 'SQL', 'SQL');
11 COMMIT;
12
13 SELECT *
14 FROM chaines
15 WHERE NomChar4 LIKE NomChar10;
```

# Recherche de base : LIKE

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

```
1 CREATE TABLE chaines
2 (
3     NomChar4      CHAR(4),
4     NomChar10     CHAR(10),
5     NomVarChar4   VARCHAR(4),
6     NomVarChar10  VARCHAR(10)
7 );
8
9 INSERT INTO chaines
10 VALUES ('SQL', 'SQL', 'SQL', 'SQL');
11 COMMIT;
12
13 SELECT *
14 FROM chaines
15 WHERE NomVarChar4 LIKE NomVarChar10;
```

- **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

```
1 SELECT nom  
2 FROM employes  
3 WHERE numchef IS NULL;  
4  
5 SELECT nom  
6 FROM employes  
7 WHERE numchef is not NULL;
```

# Recherche de base : NULL

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

```
1 SELECT nom,  
2     COALESCE(numchef, 'Président') AS Chef  
3 FROM employes;
```

# Recherche de base : NULL

Tester si une valeur a été modifiée

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

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

## Union

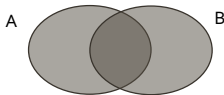
```
1 A UNION [ALL] B
2   [CORRESPONDING [BY (liste_colonne)]]
```

**liste\_colonne** :  $C_1, C_2, \dots, C_n$  ! 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

```
1 SELECT nom, prenom
2 FROM Employes
3 WHERE UPPER(commune) = 'LIEGE'
4   and sexe = 'F'
5 UNION
6 SELECT nom, prenom
7 FROM Employes
8 WHERE UPPER(commune) = 'WAREMME'
9   and sexe = 'F';
```



**Figure 2 – union**

# Recherche de base : EXCEPT

Différence

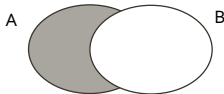
```
1 A EXCEPT [ALL] B
2 [CORRESPONDING [BY (liste_colonne)]]
```

**liste\_colonne** :  $C_1, C_2, \dots, C_n$  ! 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

```
1 SELECT NumSecu
2 FROM Employes MINUS -- en Oracle 10g
3 SELECT NumSecu
4 FROM EmpPro;
```



**Figure 3** – difference



# Recherche de base : INTERSECT

Différence

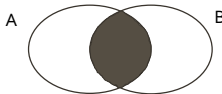
```
1 A INTERSECT [ALL] B
2   [CORRESPONDING [BY (liste_colonne)]]
```

**liste\_colonne** :  $C_1, C_2, \dots, C_n$  ! 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

```
1 SELECT NumSecu
2 FROM EmpPro
3 WHERE Numpro = 'p10346'
4 INTERSECT
5 SELECT NumSecu
6 FROM EmpPro
7 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

```
1  -- pas de résultat : un même champ ne peut avoir 2
   valeurs différentes dans un même tuple
2  SELECT NumSecu
3  FROM EmpPro
4  WHERE Numpro = 'p10346'
5     AND NumPro = 'p10349';
6
7  -- cela représente l'union et non l'intersection !
8  SELECT NumSecu
9  FROM EmpPro
10 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');
```

- 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

```
1 SELECT COALESCE(NumDep, 'Département inconnu')
2 FROM Employes
3 WHERE UPPER(sexe) = 'F'
4 MINUS
5 SELECT COALESCE(NumDep, 'Département inconnu')
6 FROM Employes
7 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

```
1 SELECT COALESCE (NumDep, 'Département inconnu')  
2 FROM Employes  
3 WHERE UPPER(sexe) = 'F'  
4 INTERSECT  
5 SELECT COALESCE (NumDep, 'Département inconnu')  
6 FROM Employes  
7 WHERE UPPER(sexe) = 'M';
```



# Recherche de base : Exercices

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

```
1 SELECT NumDep
2 FROM Employes
3 WHERE UPPER(Commune) = 'LIEGE'
4 UNION
5 SELECT NumDep
6 FROM Employes
7 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

```
1 SELECT NumSecu
2 FROM Employes
3 MINUS
4 SELECT COALESCE (NumChef, 'XXX')
5 FROM Employes;
```

## **Recherche de base avec jointure**

---

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

## 1. Introduction

## 2. Recherche de base

## 3. Recherche de base avec jointure

### 3.1 Jointure prédicative ou jointure manuelle

### 3.2 Jointure interne non-equijointure

### 3.3 Auto-jointure

### 3.4 Jointure naturelle

### 3.5 Jointure interne

### 3.6 Jointure externe

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

3.7 Jointure d'union

3.8 Remarque : clause FROM

3.9 Exercices

4. Expressions SQL

5. Tri

6. Groupement de lignes

7. Sélections imbriquées

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

8. Utilisation de "EXISTS"

9. Mise à jour des données

# Recherche de base avec jointure : Jointure prédicative ou jointure manuelle

Jointure prédicative ou jointure manuelle

```
1 SELECT Nom, NomPro
2 FROM Employes,
3     EmpPro,
4     Projets
5 WHERE Employes.NumSecu = EmpPro.NumSecu
6 AND EmpPro.NumPro = Projets.NumPro;
```

# Recherche de base avec jointure : Jointure prédicative ou jointure manuelle

Jointure prédicative ou jointure manuelle

```
1 SELECT Nom, NomPro, NomDep
2 FROM Employes,
3     EmpPro,
4     Projets,
5     Departements
6 WHERE Employes.NumSecu = EmpPro.NumSecu
7     AND EmpPro.NumPro = Projets.NumPro
8     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"

```
1 SELECT E2.Nom
2 FROM Employes E1
3         JOIN Employes E2
4         ON E2.Bareme > E1.Bareme
5 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

```
1 SELECT E1.Nom, ' a pour chef ', E2.Nom
2 FROM Employees E1,
3      Employees E2 -- même table
4 WHERE COALESCE(E1.NumChef, E1.NumSecu)
5          = E2.NumSecu
6 ORDER BY E2.Nom;
```

# Recherche de base avec jointure : Auto-jointure

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

```
1  SELECT Ep1.NumSecu
2  FROM EmpPro Ep1, EmpPro Ep2
3  WHERE Ep1.NumSecu = Ep2.NumSecu
4         AND Ep1.NumPro = 'p10346'
5         AND Ep2.NumPro = 'p10349';
6
7
8  SELECT NumSecu FROM EmpPro WHERE NumPro = 'p10346'
9  INTERSECT
10 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<sup>4</sup>.
- 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.

```
1 SELECT Nom, NumPro
2 FROM Employes
3      NATURAL JOIN EmpPro;
```

# Recherche de base avec jointure : Jointure interne

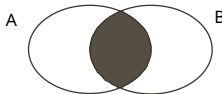
Afficher le nom des employés ainsi que le numéro des projets auxquels ils sont affectés

```
1  SELECT Nom, NumPro
2  FROM Employes
3      INNER JOIN EmpPro
4          USING (NumSecu);
5
6
7  SELECT Nom, NumPro
8  FROM Employes
9      INNER JOIN EmpPro
10         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

```
1 SELECT Nom, NomPro AS NomProjet  
2 FROM Employes  
3       INNER JOIN EmpPro USING (NumSecu)  
4       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

```
1 SELECT colonnes
2 FROM TableGauche
3   [RIGHT OUTER | LEFT OUTER | FULL OUTER] JOIN
4   TableDroite ON condition
5 [WHERE prédicat] ;
```



**Figure 6** – Jointure externe gauche



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

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

**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.

```
1 SELECT Employes.numsecu, nom
2 FROM Employes
3     LEFT OUTER JOIN EmpPro
4         ON Employes.NumSecu = EmpPro.
           NumSecu
5 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.

```
1 SELECT Employes.numsecu, nom
2 FROM Employes
3         LEFT OUTER JOIN EmpPro
4                 ON Employes.NumSecu = EmpPro.
                   NumSecu
5 WHERE EmpPro.NumSecu IS NULL;
6
7 SELECT NumSecu
8 FROM Employes MINUS -- en Oracle 10g
9 SELECT NumSecu
10 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

```
1 SELECT EmpPro.*, NULL, NULL, NULL  
2 FROM EmpPro  
3 UNION ALL  
4 SELECT NULL, NULL, NULL, Departements.*  
5 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

```
1 SELECT NomPro
2 FROM (SELECT *
3         FROM Employes
4             NATURAL JOIN EmpPro
5         WHERE UPPER(nom) = 'DE NIRO')
6         JOIN Projets USING (NumPro);
```



## 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 ?)

```
1  -- Manuelle :
2  SELECT Nom, NomDep
3  FROM Employes,
4       Departements
5  WHERE Employes.NumDep = Departements.NumDep;
6
7  -- Interne :
8  SELECT Nom, NomDep
9  FROM Employes
10         INNER JOIN Departements
11         ON (Employes.NumDep = Departements.
12             NumDep);
```

# Recherche de base avec jointure : Exercices ii

```
13 -- OU
14 SELECT Nom, NomDep
15 FROM Employes
16         INNER JOIN Departements
17                 USING (NumDep);
18
19 -- Naturelle :
20 SELECT Nom, NomDep
21 FROM Employes
22         NATURAL JOIN Departements;
23
24 -- 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

```
1 SELECT E1.Nom, E2.Nom, E1.Commune
2 FROM Employes E1,
3     Employes E2
4 WHERE E1.Commune = E2.Commune
5 AND E1.Nom < E2.Nom
6 ORDER BY E1.Nom;
```

# Recherche de base avec jointure : Exercices

Rechercher le nom des départements qui possèdent le même responsable

```
1 SELECT D1.NomDep, D2.NomDep
2 FROM Departements D1
3         INNER JOIN Departements D2 USING (NumSecu)
4 WHERE D1.NomDep < D2.NomDep;
```

# Recherche de base avec jointure : Exercices

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

```
1 SELECT NomPro
2 FROM Projets
3     LEFT JOIN EmpPro USING (NumPro)
4 WHERE NumSecu IS NULL;
```

# Recherche de base avec jointure : Exercices

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

```
1 SELECT Nom
2 FROM Employes
3         LEFT JOIN EmpPro USING (NumSecu)
4 WHERE NumPro IS NULL;
```

# Expressions SQL

---



# Table des matières de la section : Expressions SQL i

1. Introduction

2. Recherche de base

3. Recherche de base avec jointure

4. Expressions SQL

4.1 Enrichir les clauses

4.2 Expressions numériques

4.3 Expressions caractères

4.4 Opérateur CASE

4.5 Expression de dates et temps

## Table des matières de la section : Expressions SQL ii

4.6 Temps et environnements distribués

4.7 Expressions d'intervalles

4.8 L'opérateur OVERLAPS

5. Tri

6. Groupement de lignes

7. Sélections imbriquées

8. Utilisation de "EXISTS"

9. Mise à jour des données

# 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

```
1 condition_de_base ::=  
2   expression oper_comp expression  
3  
4 oper_comp ::=   = | <> | < | <= | > | >=  
5  
6 expression ::= expression_numérique  
7   | expression_caractère  
8   | expression_date_temps  
9   | expression_intervalle
```

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

```
1 expression_numérique ::=  
2   terme | terme { + | - } terme  
3  
4 terme ::= facteur | terme { * | / } facteur  
5  
6 facteur ::= [+ | -] constante  
7   | [+ | -] nom_colonne  
8   | [+ | -] fonction_de_calcul  
9   | [+ | -] fonction_de_conversion  
10  | (expression_numérique)
```

# Expressions SQL : Expressions numériques

## Expressions numériques

```
1 fonction_de_calcul ::=
2   COUNT()
3   | AVG([ALL|DISTINCT] expression)
4   | MAX([ALL|DISTINCT] expression)
5   | MIN([ALL|DISTINCT] expression)
6   | SUM([ALL|DISTINCT] expression)
7   | COUNT([ALL|DISTINCT] expression)
8
9 fonction_de_conversion ::=
10  CAST(expression AS type_de_donnée |
11     domaine)
```

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

- Dans la littérature anglo-saxonne, 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

```
1 SELECT COUNT(NumSecu) AS NB,  
2     AVG(Bareme)      AS Moyenne,  
3     MIN(Bareme)      AS "Bareme Min",  
4     MAX(Bareme)      AS "Bareme Max"  
5 FROM Employes  
6 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

```
1 SELECT Nom, Prenom, Bareme
2 FROM Employes
3 WHERE Bareme = (SELECT MIN(Bareme) FROM Employes)
4 UNION
5 SELECT Nom, Prenom, Bareme
6 FROM Employes
7 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

```
1 SELECT E1.Nom, E1.Prenom, E1.Bareme
2 FROM Employes E1,
3     (SELECT MIN(Bareme) AS BaremeMin FROM Employes) E2,
4     (SELECT MAX(Bareme) AS BaremeMax FROM Employes) E3
5 WHERE E1.bareme = E2.BaremeMin
6     OR E1.bareme = E3.BaremeMax;
```

# Expressions SQL : Expressions caractères

## Expressions caractères

```
1 expression_caractère ::=  
2 'chaîne_constante'  
3 | nom_colonne  
4 | UPPER (expression_caractère)  
5 | LOWER (expression_caractère)  
6 | CHARACTER_LENGTH (expression_caractère)  
7 | USER  
8 | CAST (expression AS type_de_donnée|domaine)  
9 | SUBSTRING (expression_caractère FROM début FOR long)  
10 | expression_caractère {||expression_caractère }  
11 | POSITION(expression_caractère IN expression_caractère)  
12 | TRIM(ltb, [pad,] FROM expression_caractère)
```

- **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.

- **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

```
1 CASE expression
2   WHEN valeur1 THEN resultat1
3   WHEN valeur2 THEN resultat2
4   ...
5   [ ELSE resultatn ]
6 END
7 ou
8 CASE
9   WHEN condition1 THEN resultat1
10  WHEN condition2 THEN resultat2
11  ...
12  [ ELSE resultatn ]
13 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.

```
1  SELECT nom,  
2      (CASE  
3          WHEN bareme >= 90000 THEN '>= 90000'  
4          WHEN bareme >= 80000 THEN 'entre 80000 et  
5              89999'  
6          WHEN bareme >= 70000 THEN 'entre 70000 et  
7              79999'  
8          ELSE '< 70000'  
9          END) AS bareme,  
10     (CASE sexe  
11         WHEN 'F' THEN 'Féminin'  
12         ELSE 'Masculin'  
13         END) AS genre  
14 FROM Employes  
15 WHERE UPPER(nom) LIKE '_E%';
```



# Expressions SQL : Expression de dates et temps

## Expression de dates et temps

```
1 expression_date_temps ::=  
2     constante  
3   | nom_colonne  
4   | CAST ( expression AS type_de_donnée | domaine)  
5   | CURRENT_DATE  
6   | CURRENT_TIME [ précision ]  
7   | CURRENT_TIMESTAMP [ précision ]  
8   | 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

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

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

```
1 EXTRACT ({YEAR | MONTH | DAY | HOUR |  
2 MINUTE | SECOND}  
3 FROM [datetime_value_expression |  
4 interval_value_expression])
```

# Expressions SQL : Expression de dates et temps

## Exemple CAST

```
1  CAST ('10/11/2017' AS DATE)
2  --Résultat : 10/11/2017 00:00:00
3
4  CAST ('10/11/2017' AS TIMESTAMP)
5  --Résultat : 10/11/2017 00:00:00.000000
6
7  CAST ('01 00:00:00' AS INTERVAL DAY TO SECOND)
8  --Résultat : +01 00:00:00.000000
9
10 CAST('01-11' AS INTERVAL YEAR TO MONTH)
11 --Résultat : +01-11
```

# Expressions SQL : Expression de dates et temps

Redéfinir le format des dates

```
1 ALTER SESSION
2   SET NLS_DATE_FORMAT = 'DD/MM/YYYY HH24:MI:SS';
3 ALTER SESSION
4   SET NLS_TIMESTAMP_FORMAT = 'DD/MM/YYYY HH24:MI:SS.FF';
5
6 SELECT TO_DATE ('10/11/2017', 'DD/MM/YYYY') FROM DUAL;
7 --Résultat : 10/11/2017 00:00:00
8
9 SELECT TO_TIMESTAMP ('10/11/2017', 'DD/MM/YYYY') FROM
10    DUAL;
11 --Résultat : 10/11/2017 00:00:00.000000
```

- 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



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
<pre>SET TIME ZONE -8:00  ! Il est 10 heures  INSERT INTO T (..., debut, ...) VALUES (...,          TIME '10:00:00', ...);</pre>	<pre>TIME '18:00:00'</pre>	<pre>SET TIME ZONE +2:00  SELECT debut FROM T;  TIME '20:00:00'</pre>

**Figure 10** – Temps et environnements distribués

# Expressions SQL : Expressions d'intervalles

1 <sup>er</sup> opérande	Opérateur	2 <sup>ème</sup> 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

```
1  SELECT nom,  
2      DateNais,  
3      EXTRACT(YEAR FROM CURRENT_DATE)  
4          - EXTRACT(YEAR FROM DateNais) Age  
5  FROM Employes;  
6  
7  SELECT nom,  
8      DateNais,  
9      (CURRENT_DATE - DateNais) YEAR TO MONTH Age  
10 FROM Employes;  
11  
12  
13 SELECT nom,  
14      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 !

- 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**

---

- 1. Introduction
- 2. Recherche de base
- 3. Recherche de base avec jointure
- 4. Expressions SQL
- 5. Tri**
  - 5.1 Syntaxe**
- 6. Groupement de lignes



## Table des matières de la section : Tri ii

7. Sélections imbriquées

8. Utilisation de "EXISTS"

9. Mise à jour des données

Tri

```
1 instruction_de_sélection ::=  
2     expression_de_sélection [ clause_de_tri ] ;  
3  
4 clause_de_tri ::= ORDER BY liste_colonne_tri  
5  
6 colonne_tri ::= colonne | numéro [ ASC | DESC ]
```

- 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 supérieure à toute valeur définie. Ce choix est laissé au constructeur.

En Oracle, NULL est supérieure<sup>8</sup> à toute valeur définie.

---

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

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;
```

## **Grouperment de lignes**

---

# Table des matières de la section : Groupement de lignes i

- 1. Introduction
- 2. Recherche de base
- 3. Recherche de base avec jointure
- 4. Expressions SQL
- 5. Tri
- 6. Groupement de lignes
  - 6.1 Syntaxe
  - 6.2 L'ordre d'évaluation

## Table des matières de la section : Groupement de lignes ii

6.3 Exemples

7. Sélections imbriquées

8. Utilisation de "EXISTS"

9. Mise à jour des données



# Groupement de lignes : Syntaxe

## Groupement de lignes

```
1 expression_de_sélection ::=  
2   SELECT [ ALL | DISTINCT ] clause_de_sélection  
3   FROM liste_table_ref  
4   [ WHERE condition ]  
5   [ clause_grouper ]  
6   [ clause_sélection_groupes ];  
7  
8 clause_grouper ::= GROUP BY liste_colonne  
9  
10 clause_sélection_groupes ::= HAVING condition
```

- 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**

- **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

```
1 SELECT NumDep, COUNT(*) NbEmp
2 FROM Employes
3 GROUP BY NumDep;
4
5 --Mieux :
6 SELECT COALESCE(NumDep, 'Inconnu') NrDep,
7         COUNT(*) NbEmp
8 FROM Employes
9 GROUP BY NumDep;
```

# Grouperment 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

```
1 SELECT nom, prenom, SUM(Heures) NbHeures
2 FROM Employes JOIN EmpPro USING (NumSecu)
3 GROUP BY NumSecu;
4 -- Ceci Provoque une erreur
5 -- Tous les champs qui apparaissent dans le SELECT
   doivent apparaître dans le GROUP BY
6
7 SELECT nom, prenom, SUM(Heures) NbHeures
8 FROM Employes JOIN EmpPro USING (NumSecu)
9 GROUP BY NumSecu, nom, prenom
10 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

```
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) < 40
6 ORDER BY 3;
7 -- 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

```
1 SELECT MIN(SUM(heures)) NbHeures  
2 FROM EmpPro  
3 GROUP BY NumSecu;
```

# Groupement de lignes : Exemples

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

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

Provoque une erreur à cause du **MIN(SUM(heures))** Voir la solution au slide 116.

# Groupement de lignes : Exemples

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

```
1 CREATE
2 OR REPLACE VIEW Temp AS (
3     SELECT Nom, Prenom, SUM(Heures) NbHeures
4     FROM Employes JOIN EmpPro USING (NumSecu)
5     GROUP BY NumSecu, Nom, Prenom);
6
7 SELECT *
8 FROM Temp
9 WHERE NbHeures = (SELECT MIN(NbHeures)
10                  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

```
1 SELECT MAX (AVG(Bareme))
2 FROM Employes;
3 --Provoque une erreur !!!
4
5 SELECT MAX(AVG(Bareme)) AS MaxAvg
6 FROM Employes
7 GROUP BY NumDep;
8 --OU
9 SELECT MAX(Temp.Moyenne)
10 FROM (SELECT AVG(Bareme) AS Moyenne
11        FROM Employes
12        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

```
1 SELECT NumDep, MAX(AVG(Bareme)) AS MaxAvg
2 FROM Employes
3 GROUP BY NumDep;
4 -- Provoque une erreur ! MAX(AVG(Bareme)) - ORA-00937
5
6 SELECT NumDep, AVG(Bareme) AS MaxAvg
7 FROM Employes
8 GROUP BY NumDep
9 HAVING AVG(Bareme) = (SELECT MAX(AVG(Bareme))
10                      FROM Employes
11                      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

```
1 instruction_de_sélection ::=  
2     expression_de_sélection [ clause_de_tri ] ;  
3 expression_de_sélection ::=  
4     SELECT clause_de_sélection FROM liste_table_ref  
5     [ WHERE condition ]  
6     [ clause_grouper ]  
7     [ clause_sélection_groupes ]  
8  
9 condition ::=  
10    [ NOT ] condition_élémentaire  
11    | condition AND | OR condition  
12    | ( condition )  
13 condition_élémentaire ::=  
14    condition_de_base | condition_between |  
    condition_in | condition_like |  
    condition_null | condition_all_any |  
    condition_exists
```

**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

```
1 SELECT Nom, Prenom
2 FROM Employes
3 WHERE NumSecu IN (SELECT NumSecu
4                     FROM EmpPro
5                     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

```
1 SELECT NumSecu
2 FROM Employes
3 WHERE NumSecu NOT IN (SELECT NumSecu
4                        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);
```

## 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

```
1 SELECT NomDep
2 FROM Departements
3 WHERE NumDep NOT IN
4     (SELECT COALESCE(NumDep, 'Sans Numéro')
5      FROM Employes
6      WHERE Bareme < 60000);
```



# Sélections imbriquées : Exemples

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

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

`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 `sélection_une_colonne`
- Le résultat du `sélection_une_colonne` est vide

`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 `sélection_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 `sélection_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

```
1 SELECT NumSecu
2 FROM Employes
3 WHERE Bareme > ALL (SELECT Bareme
4                     FROM Employes
5                     WHERE NumDep = 'd00002');
```



# Sélections imbriquées : Exemples

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

```
1 SELECT Nom
2 FROM Employes
3 WHERE Bareme >=
4         ALL (SELECT Bareme
5              FROM Employes
6              WHERE bareme IS NOT NULL);
```

# Sélections imbriquées : Exemples

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

```
1 CREATE
2 OR REPLACE VIEW Temp AS (
3 SELECT Nom, Prenom, SUM(heures) NbHeures
4 FROM Employes JOIN EmpPro USING (NumSecu)
5 GROUP BY NumSecu, Nom, Prenom);
6
7 SELECT *
8 FROM Temp
9 WHERE NbHeures = (SELECT MIN(NbHeures)
10                  FROM Temp);
```

# Sélections imbriquées : Exemples

Exemple : Afficher 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      ALL (SELECT SUM(Heures)
7           FROM EmpPro
8           GROUP BY NumSecu)
9 ORDER BY 3;
```

## 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), (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

```
1  SELECT Nom
2  FROM Employes E1
3  WHERE NumDep <> (SELECT NumDep
4                   FROM Employes
5                   WHERE NumSecu = E1.NumChef);
6
7  SELECT Nom
8  FROM Employes E1
9  WHERE COALESCE(NumDep, 'X') <>
10         (SELECT COALESCE (NumDep, 'X')
11          FROM Employes
12          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

```
1 SELECT Nom, Prenom
2 FROM Employes E1
3 WHERE Bareme >
4     (SELECT AVG(bareme)
5      FROM Employes E2
6      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

```
1  SELECT Nom, Prenom
2  FROM Employes
3  WHERE 0 <
4         (SELECT COUNT(*)
5          FROM EmpPro
6          WHERE heures = 10
7                 AND NumPro = 'p10349'
8                 AND EmpPro.NumSecu = Employes.NumSecu);
9  -- ou avec une jointure
10 select nom, PRENOM
11 from EMPLOYES
12         inner join EMPPRO E on EMPLOYES.NUMSECU = E.
13                                NUMSECU
14 where HEURES = 10
15        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

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

## **Utilisation de "EXISTS"**

---

# Table des matières de la section : Utilisation de "EXISTS" i

1. Introduction
2. Recherche de base
3. Recherche de base avec jointure
4. Expressions SQL
5. Tri
6. Groupement de lignes
7. Sélections imbriquées

## Table des matières de la section : Utilisation de "EXISTS" ii

### 8. Utilisation de "EXISTS"

#### 8.1 Tester l'existence de données

#### 8.2 Exemples

### 9. Mise à jour des 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

```
1 SELECT Nom, Prenom
2 FROM Employes
3 WHERE EXISTS
4     (SELECT * -- Constante arbitraire
5      FROM EmpPro
6      WHERE NumSecu = Employes.NumSecu -- Ici il y a de la
7          corrélation
8      AND Heures = 10
9      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
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  $\implies$  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

```
1 SELECT Nom
2 FROM Employes
3 WHERE bareme = (SELECT MAX(bareme)
4                  FROM Employes);
```

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

1. Introduction
2. Recherche de base
3. Recherche de base avec jointure
4. Expressions SQL
5. Tri
6. Groupement de lignes
7. Sélections imbriquées

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

## ii

8. Utilisation de "EXISTS"

9. Mise à jour des données

9.1 INSERT

9.2 UPDATE

9.3 DELETE

# Mise à jour des données : INSERT

INSERT

```
1 instruction_d_ajout ::=  
2  
3 INSERT INTO nom_table [ ( liste_colonne ) ]  
4     VALUES ( liste_valeur )  
5         | 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);
```

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

```
1  INSERT INTO employes
2      (numsecu, nom, prenom, numdep, bareme, numchef)
3  VALUES ('888888', 'DUBOIS', 'Luc',
4          (SELECT numdep
5           FROM employes
6           WHERE numdep IS NOT NULL
7           GROUP BY numdep
8           HAVING COUNT(*) <= ALL
9                (SELECT COUNT(*)
10                 FROM employes
11                 WHERE numdep IS NOT NULL
12                 GROUP BY numdep)),
13          (SELECT AVG(bareme) FROM employes),
14          (SELECT numsecu FROM employes WHERE nom = '
           HOFFMAN'));
```

# Mise à jour des données : UPDATE

## UPDATE

```
1 instruction_de_modification ::=  
2 UPDATE nom_table  
3 SET liste_colonne_valeur  
4 [ WHERE condition ] ;  
5  
6 liste_colonne_valeur ::=  
7   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

```
1 instruction_de_suppression ::=  
2  
3 DELETE FROM nom_table [ WHERE condition ] ;
```

# Mise à jour des données : DELETE

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

```
1 DELETE
2 FROM EmpPro
3 WHERE numsecu
4         IN (SELECT numsecu
5             FROM employes
6             WHERE nom = 'CELARIE'
7                 AND prenom = 'Clémentine');
```

- Les présents diaporamas constituent un résumé du livre : DELMAL, P, *SQL2 - SQL3*, Bruxelles, De Boeck Université, 2004, 512 pages.
- Ce résumé a été étoffé d'éléments extraits de la documentation officielle d'Oracle : [Oracle Database Documentation](#), 12/02/2021.
- Ainsi que de l'ouvrage : SOUTOU, Ch, *SQL pour Oracle*, Paris, Editions Eyrolles, 2013, 642 pages.