SQL

Ch 2 – SELECT
Interroger les données 1

I. Introduction

L'ordre **SELECT** permet l'interrogation du contenu de tables d'une base de données. Le résultat renvoyé suite au traitement de la requête SQL par le SGBDR est une table temporaire (des colonnes et des lignes).

A. Syntaxe générale de l'ordre SELECT

Chaque clause de l'ordre SQL SELECT a un rôle bien précis dans la construction du résultat final.

Conventions :

- Entre [ ] : des valeurs optionnelles
- Entre { } : une liste de valeur obligatoires possibles
- De part et d'autre de | : une valeur parmi celles proposées
- Souligné : valeur par défaut (elle est souvent omise)
- En caractères MAJUSCULES : les mots clefs du langage

Syntaxe générale :

```
SELECT [ DISTINCT | ALL ] { * | liste_de_colonnes }
  FROM nom_de_table
[ WHERE critères_de_selection ]
[ GROUP BY liste_de_colonnes_de_regroupement ]
[ HAVING critères_de_selection_après_groupement ]
[ ORDER BY liste_de_colonnes_de_tri ]
[ LIMIT n,m]

;
```

- **Liste_de_colonnes**
  - Colonnes à renvoyer (opérateur relationnel « projection »)
- **Nom_de_table** :
  - Nom de la table (ou des tables) qui permet de répondre à la question
- **Critères_de_selection**
  - Expression logique permettant de restreindre le nombre de lignes récupérées dans la table (opérateur relationnel « sélection »)
- **Liste_de_colonnes_de_regroupement**:
  - Noms des colonnes sur lesquelles des sous-totaux sont calculés
- **Criteres_de_selection_apres_regroupement**:
  - Expression logique permettant de restreindre le nombre de lignes après calcul des sous-totaux
- **Liste_de_colonnes_de_tri**:
  - Nom de colonnes appartenant à la table, qui vont permettre le classement des lignes pour fournir le résultat final et critère de classement : ASC (croissant), DESC (décroissant)

B. Extrait des Tables utilisées en exemple

- **t_membre** (**id_memb**, nom, prenom, adresse, ville, dateNais, **#id_memb_parrain**)
- **t_artiste** (**id_artiste**, nom, prenom, anneeNais, paysOrigine)
- **t_oeuvre** (**id_oeuvre**, titre, annee, **#id_artiste**)
- **t_peinture** (**#id_oeuvre**, hauteur, largeur)

Soit : nom_de_table (les noms des colonnes composant la table)

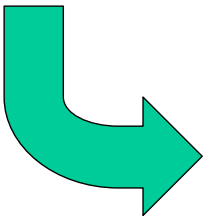
- Une colonne soulignée représente la clef primaire de la table,
- une colonne préfixée par # représente une clef étrangère (une clef primaire dans une autre table)

II. « SELECT » : projection relationnelle

L'ordre SELECT précise les noms des colonnes à retourner dans le résultat final.

table résultat de :
SELECT nom, prenom
FROM t_personnel;

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07



nom	prenom
dupont	max
durant	tim
lambert	betty
bradford	jean

Figure 1 : choisir les colonnes de la table dont on souhaite récupérer le contenu

A. Syntaxe

```
SELECT [ALL | DISTINCT] liste_de_colonnes
  FROM nom_de_table
;
```

Dans la liste de colonnes spécifiée dans la clause SELECT, chaque colonne doit être séparée de la suivante par une virgule.
Contrairement à l'algèbre relationnelle, le projection SQL n'élimine pas les doublons de lignes. La clause DISTINCT permet la suppression des lignes en double (= si les valeurs de toutes les colonnes sont identiques).
La clause ALL, appliquée par défaut, retourne toutes les lignes (avec éventuellement des doublons).

B. Exemples

Exemple : Afficher la colonne **dateNais** de la table **t_membre**

```
SELECT dateNais
    FROM t_membre;
```

Exemple : Afficher les colonnes **nom** et **prenom** de la table **t_membre**

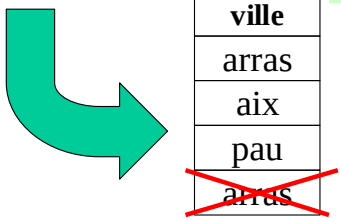
```
SELECT nom, prenom
    FROM t_membre;
```

C. Le mot clef DISTINCT : éliminer les doublons

Le mot clef DISTINCT supprime les lignes en double du résultat final.

table résultat de :
SELECT DISTINCT ville
FROM t_personnel;

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07



ville
arras
aix
pau
~~arras~~

Figure 2 : les différentes villes de la table t_personnel

Exemple : Afficher les différents prénoms de la table t_membre (supprimer les doublons)

```
SELECT DISTINCT prenom
    FROM t_membre;
```

Exemple : Afficher les différents pays d'origine et années de naissance des artistes

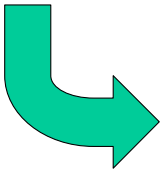
```
SELECT DISTINCT paysOrigine, annee
    FROM t_artiste;
```

D. Alias de colonne, opération de renommage : AS

Il arrive que des noms de colonnes ne soient pas significatifs ou clairs pour l'utilisateur. Il est alors possible de changer ce nom de colonne dans le résultat fourni (on parle d'alias).

table résultat de :
SELECT nom, prenom, salaire AS salaireMensuel
FROM t_personnel;

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07



nom	prenom	salaireMensuel
dupont	max	1000
durant	tim	1500
lambert	betty	1350
bradford	jean	1250

Syntaxe :

```
nom_de_colonne AS nom_d_alias
```

Exemple : Afficher la colonne **dateNais** renommée en **dateDeNaissance** (table **t_membre)**

```
SELECT dateNais AS dateDeNaissance
    FROM t_membre;
```

Remarques : Les alias de colonnes seront particulièrement utiles pour renommer les colonnes calculées, ou agrégées.

III.« FROM » : définir la table utilisée

La clause **FROM** de l'ordre SELECT définit la ou les tables à utiliser pour trouver toutes les colonnes nécessaires au traitement de la requête.

Dans ce chapitre, nous donnerons des ordres ne nécessitant l'utilisation que d'une seule table.

table « **t_personnel** »

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07

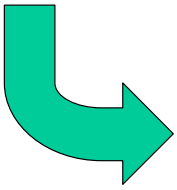
Figure 3 : définir le nom de la table à utiliser pour pouvoir exécuter la requête : les colonnes associées à cette table seront accessibles

IV. « WHERE » : sélection relationnelle

table résultat de :

```
SELECT *
  FROM t_personnel
  WHERE numero = 2;
```

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	24/04/07
4	bradford	jean	arras	1250	30/09/07



numero	nom	prenom	ville	salaire	entree
2	durant	tim	aix	1500	15/03/07

Figure 4 : choisir les lignes qui nous intéressent (à retenir)

A. Syntaxe

```
SELECT *
  FROM nom_de_table
```

```
  WHERE critères_de_selection ;
```

La clause **WHERE** (**restriction sur les lignes**) permet de ne retenir que les lignes de la table répondant à la condition exprimée dans les critères de sélection.

Les critères de sélection correspondent à une expression logique qui est évaluée pour chacune des lignes lues par le SGBD : si l'expression est vraie pour la ligne, celle-ci est retenue.

B. Construction des critères de sélection

Comme dans la plupart des langages, les critères de sélection (expressions logiques) sont construits à l'aide des opérateurs relationnels (=, >, <, >=, <=, <>) et des opérateurs logiques (AND, OR, NOT).
La norme SQL prévoit également des opérateurs spécifiques : LIKE, IN, BETWEEN.

Les constantes littérales utilisées dans les comparaisons sont de 3 types :

- Les littéraux numériques (entiers, réels) : exemple : 1, 2, 3, -100, 12563
- Les littéraux chaînes de caractères : entre guillemets (simples ou doubles selon les SGBD) "arras"
- Les valeurs littérales exprimant des dates : entre guillemets (simples ou doubles selon les SGBD) sous forme "jj/mm/aaaa" (parfois "aaaa/mm/jj", selon le SGBD) : par exemple "01/12/2007", "31/12/2007"

1. Opérateurs de comparaison : =, >, >=, <, <=, <>, !=

Exemple : Afficher toutes les colonnes : choisir les lignes pour lesquelles la valeur de la colonne **id_memb** est inférieure à 10

```
SELECT * FROM t_membre
  WHERE id_memb < 10;
```

Exemple : Afficher toutes les colonnes de la table t_membre : choisir les lignes pour lesquelles la valeur de la colonne **dateNais** est supérieure ou égal au 1er janvier 1985

```
SELECT * FROM t_membre
  WHERE dateNais >= '01/01/1985';
```

Exemple : Afficher toutes les colonnes de la table t_artiste : choisir les lignes pour lesquelles la valeur de la colonne **paysOrigine** est égale à 'Italie'

```
SELECT * FROM t_artiste
  WHERE paysOrigine = 'italie';
```

2. Comparaison avec une liste de valeurs : IN (…,…), NOT IN (…,…)

L'opérateur **IN** permet la comparaison de la valeur d'une colonne avec une liste de valeurs données : **la colonne doit être égale à l'une des valeurs pour que la condition soit vraie**.

```
colonne [NOT] IN (valeur1, valeur2, . . ., valeurN)
```

L'opérateur **IN** est équivalent à :

```
colonne = valeur1
OR colonne = valeur2
 . . .
OR colonne = valeurN)
```

table résultat de :
SELECT numero,nom, prenom
FROM t_personnel
WHERE ville **IN** ('arras' , 'aix');

table résultat de :
SELECT numero, nom, prenom
FROM t_personnel
WHERE numero **IN** (1 , 3 , 5);

	numero	nom	prenom	ville	salaire	entree	
O	1	dupont	max	arras	1000	01/01/07	O
O	2	durant	tim	aix	1500	15/03/07	N
N	3	lambert	betty	pau	1350	24/04/07	O
O	4	bradford	jean	arras	1250	30/09/07	N

numero	nom	prenom
1	dupont	max
2	durant	tim
4	bradford	jean

numero	nom	prenom
1	dupont	max
3	lambert	betty

Exemple : Afficher toutes les colonnes de la table **t_artiste** : choisir les lignes pour lesquelles la colonne **paysOrigine** est l'une des valeurs 'France' ou 'Italie'.

```
SELECT * FROM t_artiste
  WHERE paysOrigine IN ('france', 'italie');
```

Exemple : Afficher toutes les colonnes de la table **t_artiste** : choisir les lignes pour lesquelles la colonne **paysOrigine** N'est PAS l'une des valeurs 'France' ou 'Italie'.

```
SELECT * FROM t_artiste
  WHERE paysOrigine NOT IN ('france', 'italie');
```

3. Concordance avec des modèles : LIKE

L'opérateur **LIKE** permet de comparer la valeur d'une colonne de type chaîne de caractères avec un modèle. Ce modèle est constitué d'un texte dans lequel peuvent être positionnés des caractères de substitution (ou caractères jokers ou caractères omnibus) :

- Le caractère **%** (ou *) : remplace plusieurs caractères
- Le caractère _ (ou ?) : remplace un seul caractère

Ainsi, on peut effectuer les types de comparaison suivants :

- La valeur d'une colonne
  - **commence par** 'xx' : nom_colonne LIKE 'xx%'
  - **contient** 'xx' : nom_colonne LIKE '%xx%'
  - **se termine par** 'xx' : nom_colonne LIKE '%xx'
  - **contient un 'x' au caractère 2** : nom_colonne LIKE '_x%'
  - **contient un 'x' en avant dernier caractère** : nom_colonne LIKE '%x_%'
  - **est longueu de n caractères**
    - par exemple 2 **:** nom_colonne LIKE '__'

Exemple : Afficher toutes les colonnes de la table **t_artiste** : choisir les lignes pour lesquelles la colonne **paysOrigine** COMMENCE PAR 'es', (=avec n'importe quoi après)

```
SELECT * FROM t_artiste
```

```
  WHERE paysOrigine LIKE 'es%';
```

4. Comparaison avec un intervalle de valeurs : BETWEEN … AND …, NOT BETWEEN … AND …

L'opérateur **BETWEEN** permet de vérifier que la valeur d'une colonne est comprise entre 2 bornes (une borne inférieure et une borne supérieure, toutes 2 incluses).
L'opérateur **BETWEEN** permet la comparaison des valeurs numériques, mais aussi des dates et chaînes de caractères (l'ordre alphabétique est utilisé)

```
colonne [NOT] BETWEEN valeurMini AND valeurMaxi
```

L'opérateur **BETWEEN** est équivalent à :

```
colonne >= valeurMini AND colonne <= valeurMaxi
```

table résultat de :
SELECT numero,nom, prenom
FROM t_personnel
WHERE numero **BETWEEN** 3 **AND** 12;

	numero	nom	prenom	ville	salaire	entree	
N	1	dupont	max	arras	1000	01/01/07	N
N	2	durant	tim	aix	1500	15/03/07	O
O	3	lambert	betty	pau	1350	24/04/07	O
O	4	bradford	jean	arras	1250	30/09/07	N

numero	nom	prenom
3	lambert	betty
4	bradford	jean

table résultat de :
SELECT numero, nom, prenom
FROM t_personnel
WHERE entree **BETWEEN** '01/03/2007' **AND** '30/04/2007'

numero	nom	prenom
2	durant	tim
3	lambert	betty

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la colonne **annee** est comprise entre 1800 et 1900

```
SELECT * FROM t_oeuvre
  WHERE annee  BETWEEN 1800 AND 1900;
```

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la colonne **annee** N'est PAS comprise entre 1800 et 1900

```
SELECT * FROM t_oeuvre
  WHERE annee  NOT BETWEEN 1800 AND 1900;
```

5. Utiliser des connecteurs logiques : AND, OR

Les connecteurs logiques **AND** et **OR** permettent de réaliser des combinaisons plus complexes d'expression logiques simples.

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la colonne **annee** est supérieure ou égal à 1800 ET inférieure ou égal à 1900 (= comprise entre 1800 et 1900, les bornes sont comprises)

```
SELECT * FROM t_oeuvre
  WHERE annee >= 1800 AND annee <= 1900;
```

(L'opérateur BETWEEN serait plus approprié ici…)

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la valeur de la colonne **annee** est inférieure à 1800 OU supérieure à 1900

```
SELECT * FROM t_oeuvre
  WHERE annee < 1800 OR annee > 1900;
```

(L'opérateur NOT BETWEEN serait plus approprié ici…)

6. Inverser une condition : NOT

L'opérateur **NOT** permet d'inverser l'évaluation d'une condition.

```
NOT ( expression_logique )
```

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la condition **annee** N'EST PAS comprise entre 1800 et 1900.

```
SELECT * FROM t_oeuvre
  WHERE NOT (annee >= 1800 AND annee <= 1800);
```

7. Tester les valeurs non renseignées : IS NULL, IS NOT NULL

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la valeur de la colonne **annee** n'est pas renseignée (= EST NULLE)

```
SELECT * FROM t_oeuvre
  WHERE annee IS NULL;
```

Exemple : Afficher toutes les colonnes de la table **t_oeuvre** : choisir les lignes pour lesquelles la valeur de la colonne **annee** est renseignée (= N'EST PAS NULLE)

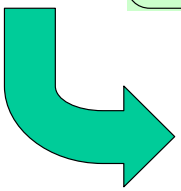
```
SELECT * FROM t_oeuvre
  WHERE annee IS NOT NULL;
```

V. « SELECT – FROM - WHERE » : choisir des colonnes et des lignes

La combinaison du choix des colonnes (SELECT) et du choix des lignes (WHERE) fournit un sous-ensemble de la table : pour les colonnes sélectionnées, seulement les valeurs des lignes choisies (« sélection » + « projection » relationnelle)

table résultat de :
SELECT nom, prenom
FROM t_personnel
WHERE numero = 2;

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07



nom	prenom
durant	tim

Figure 5 : choix de colonnes et de lignes

VI. « GROUP BY » : agrégats relationnels

Il est ensuite possible d'appliquer des fonctions d'agrégation (*de regroupement avec calculs de totaux*) aux lignes retenues, en utilisant des fonctions fournies par le langage SQL :

- **COUNT** = permet de compter le nombre de lignes sélectionnées
- **SUM** = permet d'effectuer la somme d'une colonne choisie
- **AVG** = permet d'effectuer la moyenne
- **MIN** : permet la recherche de la valeur minimale d'une colonne
- **MAX** = permet la recherche de la valeur maximale d'une colonne

Ces fonctions d'agrégations s'appliquerons :

- Au comptage des lignes d'une table : COUNT(*)
- Au comptage des colonnes renseignées d'une colonne : COUNT(nom_de_colonne)
- A des « calculs » sur les valeurs d'une colonne :
  - SUM(nom_colonne) : somme des valeurs de nom_colonne
  - AVG(nom_colonne) : moyenne des valeurs de nom_colonne
  - MIN(nom_colonne) : valeur minimale de nom_colonne (nombre, date)
  - MAX(nom_colonne) : valeur maximale de nom_colonne (nombre, date)

Ces fonctions d'agrégations fourniront :

- Soit un résultat unique calculé sur l'ensemble des lignes : totaux globaux
- Soit un résultat calculé par valeurs différentes d'une colonne : sous-totaux par.

A. Totaliser en globalité : totaux globaux

table résultat de :
SELECT COUNT(*)
FROM t_personnel;

table résultat de :
SELECT SUM(salaire)
FROM t_personnel;

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07

count
4

sum
5100

Figure 6 : totaliser en globalité sur TOUTES les lignes d'une table : compter le nombre de lignes, ou effectuer la somme des valeurs d'une colonne

1. Syntaxe

```
SELECT fonction_agregation
  FROM nom_de_table
  [ WHERE critères_de_selection ]
;
```

2. Exemples

Exemple : Afficher le nombre de lignes de la table **membre**

```
SELECT COUNT(*)
    FROM t_membre;
```

Exemple : Afficher le nombre de lignes de la table **t_membre** après avoir choisi les lignes pour lesquelles la valeur de la colonne **id_memb** est inférieure à 50

```
SELECT COUNT(*)
    FROM t_membre
    WHERE id_memb < 50;
```

Exemple : Afficher la somme des valeurs des colonnes **largeur** et **longueur** (table **t_peinture)**

```
SELECT SUM(largeur) AS LargeurTotale, SUM(longueur)
AS LongeurTotale
    FROM t_peinture;
```

3. Totaux globaux sur un sous-ensemble des lignes

Il est possible d'effectuer des totaux sur une partie des lignes seulement : seules els lignes retenues après application de la clause WHERE participeront au calcul des totaux.

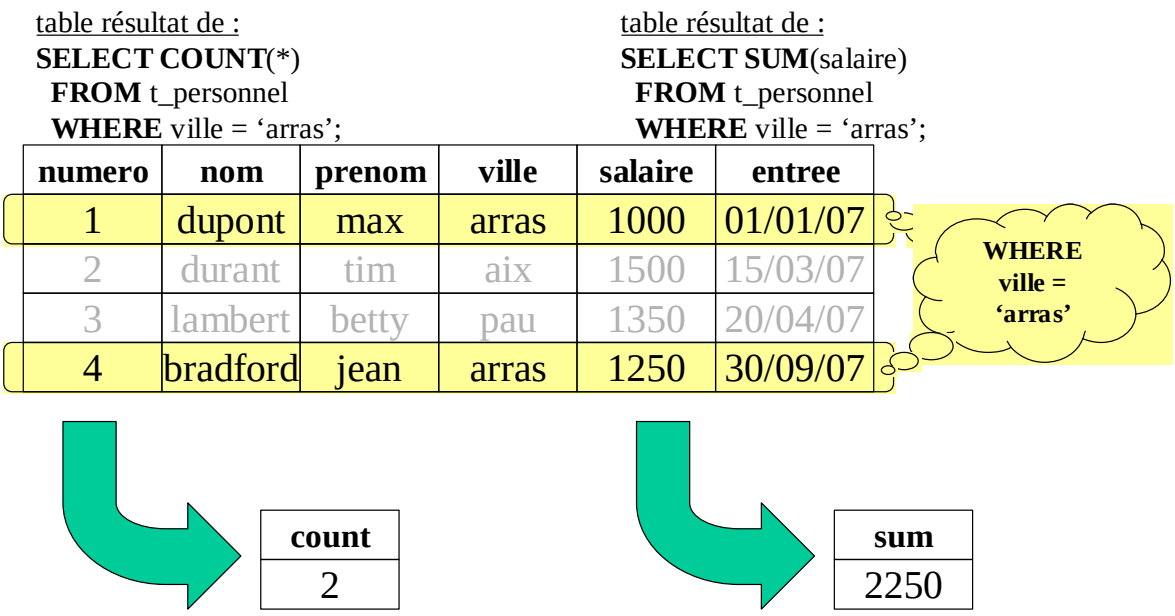
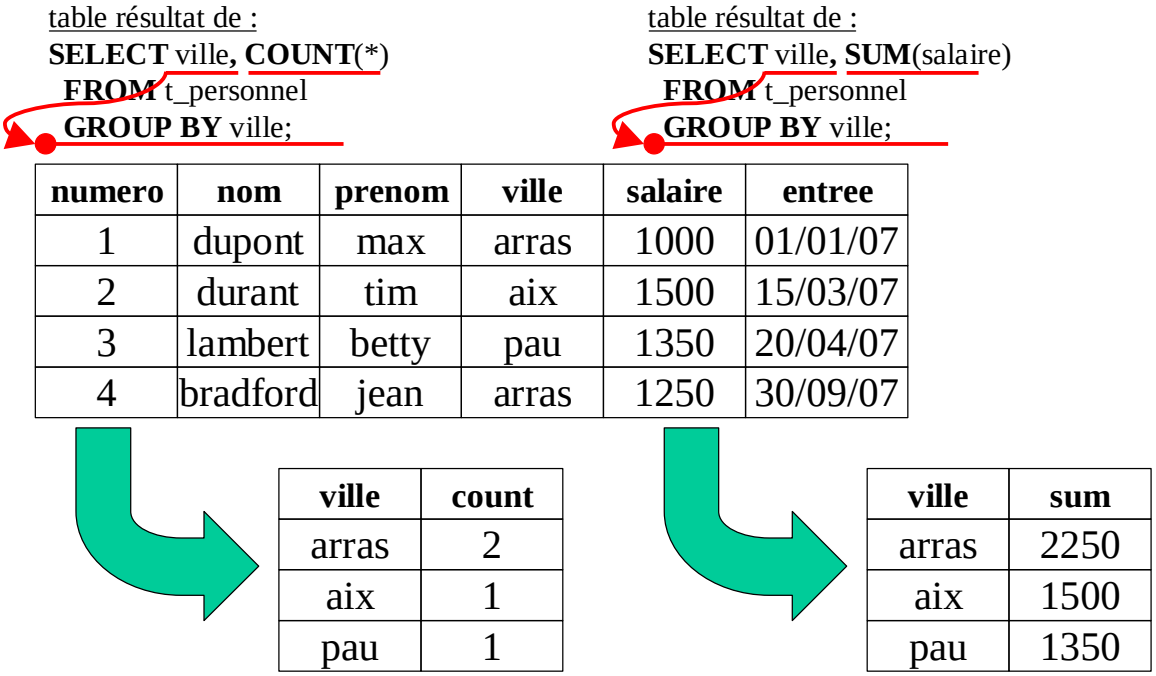


Figure 7 : totaliser en globalité sur CERTAINES lignes d'une table : compter le nombre de lignes, ou effectuer la somme des valeurs d'une colonne

B. Totaliser par (colonne(s) de regroupement) : sous-totaux par



1. Syntaxe

```
SELECT nom_de_colonne, fonction_agregation
  FROM nom_de_table
  [ WHERE critères_de_selection ]
  GROUP BY nom_de_colonne
;
```

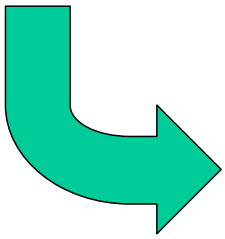
2. Exemples

La ou les colonnes de regroupement ne seront pas obligatoirement affichées : le regroupement par ville est ici réalisé, et seul les comptages et sommes sont affichées.
Sauf cas particuliers que nous étudierons plus tard, on affichera les colonnes impliquées dans les regroupements afin que ceux-ci aient un sens.

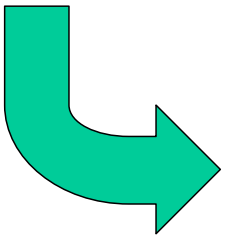
table résultat de :
```
SELECT COUNT(*)
  FROM t_personnel
  GROUP BY ville;
```

table résultat de :
```
SELECT SUM(salaire)
  FROM t_personnel
  GROUP BY ville;
```

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07



count
2
1
1



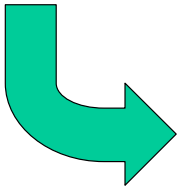
sum
2250
1500
1350

Figure 8 : calcul de sous-totaux pour chaque valeur de la colonne 'ville' : comptage des lignes par ville, somme des salaires par ville

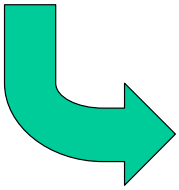
table résultat de :
```
SELECT ville, COUNT(*)
  FROM t_personnel
  GROUP BY ville;
```

table résultat de :
```
SELECT ville, SUM(salaire)
  FROM t_personnel
  GROUP BY ville;
```

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07



ville	count
arras	2
aix	1
pau	1



ville	sum
arras	2250
aix	1500
pau	1350

Figure 9 : en général, le nom des colonnes de regroupement est également pris pour être renvoyé dans le résultat final

Exemple : Afficher, pour chaque pays (par pays), le nombre de lignes de la table **t_artiste**

```
SELECT paysOrigine, COUNT(*)
    FROM t_artiste
    GROUP BY paysOrigine;
```

Il est possible d'effectuer des sous-totaux sur une partie des lignes seulement : seules les lignes retenues après restriction de la clause WHERE participeront au calcul des sous-totaux.

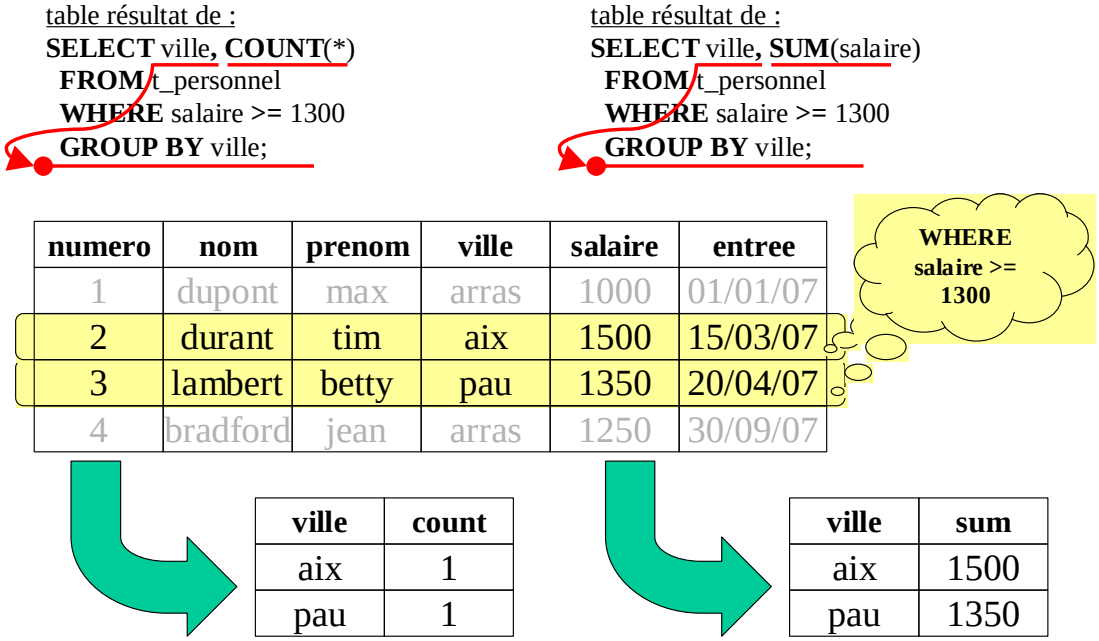


Figure 10 : effectuer des sous-totaux sur CERTAINES lignes d'une table

VII. « HAVING » : sélection relationnelle

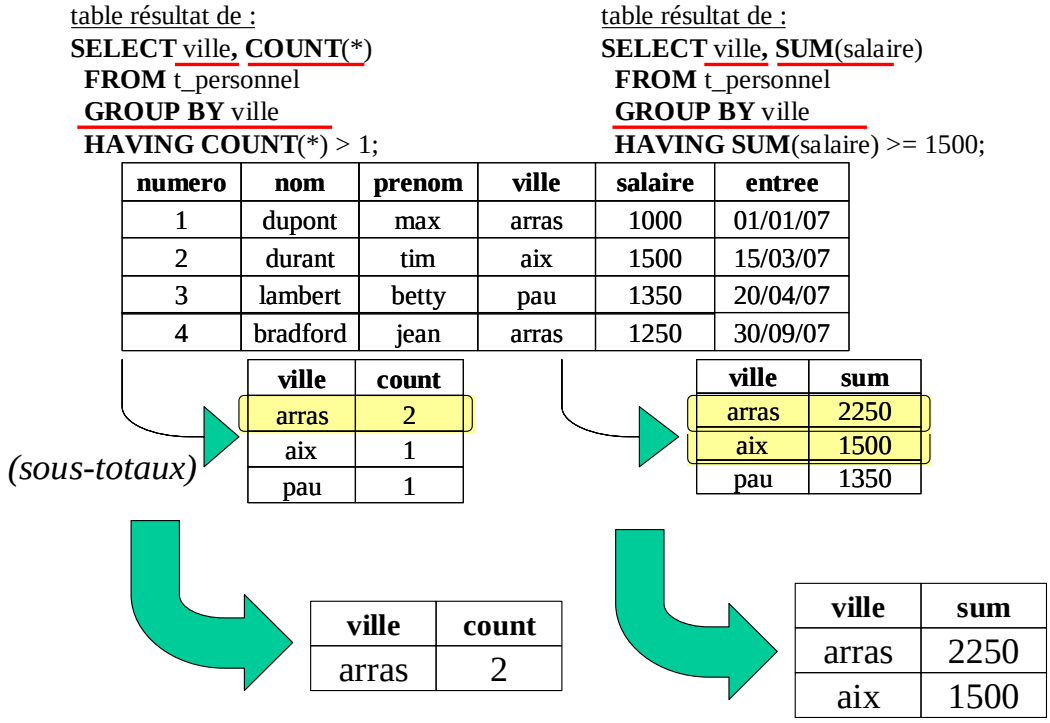


Figure 11 : sélection les totaux par ville - nombre total de lignes par ville supérieur à 1 - somme des salaires par ville supérieur ou égal à 1500

A. Syntaxe

```
SELECT colonne_regroupement, fonction_agregation
  FROM nom_de_table
  [ WHERE critères_de_selection ]
  GROUP BY colonne_regroupement
  HAVING critères_de_selection_après_regroupement
;
```

La clause **HAVING** va essentiellement utiliser des opérateurs relationnels et comparer la valeur d'une colonne agrégée à une valeur fixe (ou calculée).

B. Exemples

Exemple : Afficher le nombre de lignes de la table **t_artiste** par valeur de paysOrigine, mais ne conserver que les valeurs de paysOrigine pour lesquelles le nombre d'artistes est supérieur à 2.

```
SELECT paysOrigine, COUNT(*)
    FROM t_artiste
    GROUP BY paysOrigine
    HAVING COUNT(*) > 2;
```

Si l'agrégat avait été renommé :.

```
SELECT paysOrigine, COUNT(*) AS Nombre
    FROM t_artiste
    GROUP BY paysOrigine
    HAVING Nombre > 2;
```

VIII. « ORDER BY » : classer le résultat final

La clause ORDER BY permet le classement du résultat final. Plusieurs colonnes de classement peuvent être nommées (séparées par des virgules), avec pour chacune un critère de classement.
Les critères de classement sont :

- Le classement croissant : ASC, pour ASCending (valeur par défaut)
- Le classement décroissant : DESC, pour DESCending

A. Syntaxe

```
SELECT colonne_regroupement [, fonction_agregation]
  FROM nom_de_table
  [ WHERE critères_de_selection ]
  [ GROUP BY colonne_regroupement ]
  [ HAVING critères_de_selection_après_regroupement ]
  ORDER BY colonnes_et_critères_de_classement
;
```

table résultat de :

```
SELECT ville, SUM(salaire)
  FROM t_personnel
  GROUP BY ville
  HAVING SUM(salaire) >= 1500
  ORDER BY ville ASC;
```

numero	nom	prenom	ville	salaire	entree
1	dupont	max	arras	1000	01/01/07
2	durant	tim	aix	1500	15/03/07
3	lambert	betty	pau	1350	20/04/07
4	bradford	jean	arras	1250	30/09/07

(sous-totaux)

ville	sum
arras	2250
aix	1500
pau	1350

(filtrage sous-totaux)

ville	sum
arras	2250
aix	1500

ville	sum
aix	1500
arras	2250

B. Exemples

Exemple : Afficher la liste des membres classés par ordre alpha du nom, puis du prénom

```
SELECT nom, prenom
    FROM t_artiste
    ORDER BY nom, prenom ;
```

Exemple : Afficher le nombre de lignes de la table **t_artiste** par valeur de paysOrigine, en ne prenant que les valeurs de paysOrigine pour lesquelles le nombre d'artistes est supérieur à 2 et classer par ordre décroissant du nombre d'artistes par pays..

```
SELECT paysOrigine, COUNT(*) AS nbArtistes
    FROM t_artiste
    GROUP BY paysOrigine
    HAVING COUNT(*) > 2
    ORDER BY nbArtistes DESC;
```

IX. En résumé

On peut résumer la construction d'une requête SQL appliquée à une seule table aux choix suivants :

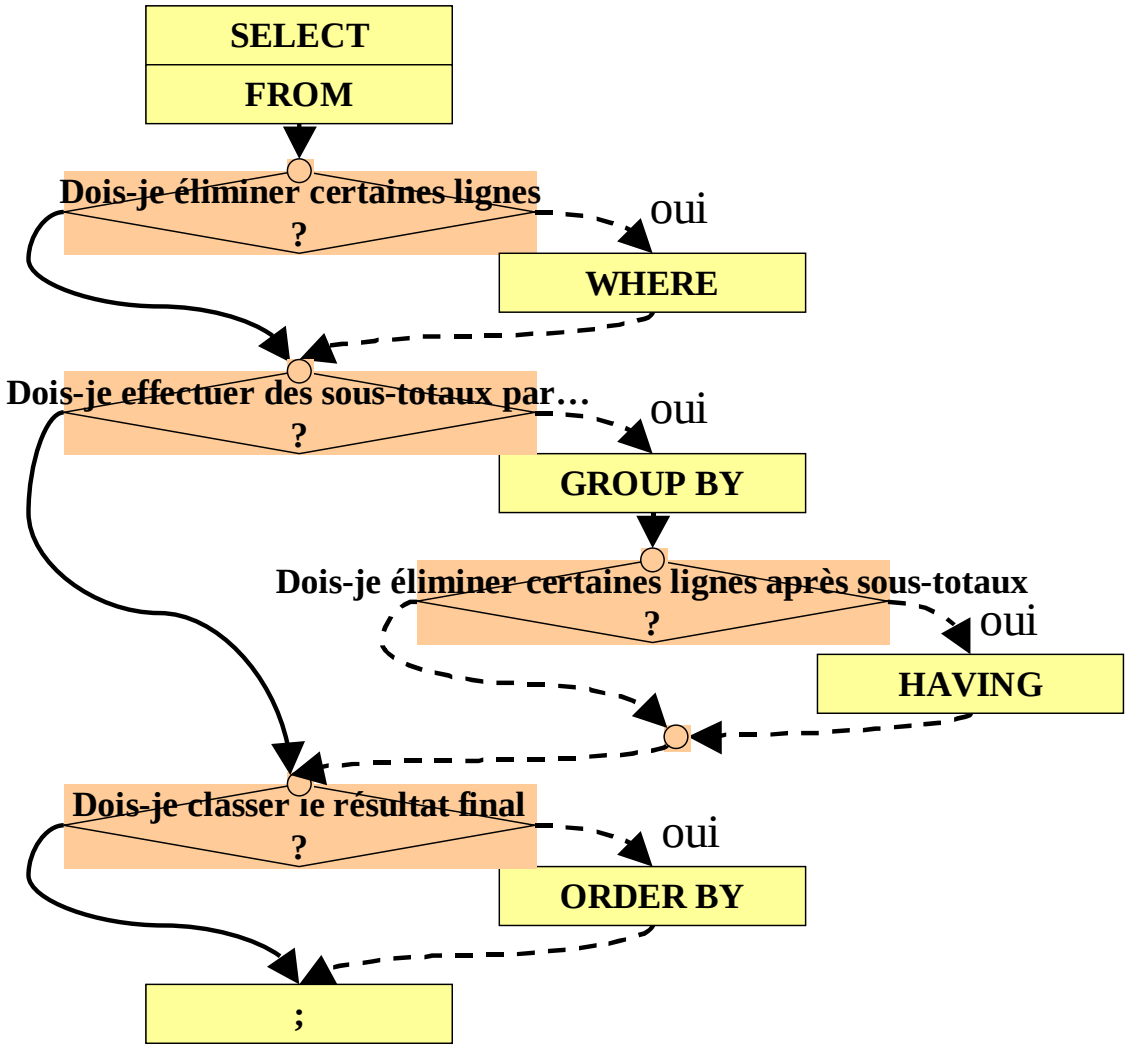


Figure 12 : construire une requête SQL

Exemple de synthèse :

```
SELECT ville, SUM(salaire)
    FROM t_personnel
    WHERE numero <= 4
    GROUP BY ville
    HAVING SUM(salaire) > 1400
    ORDER BY ville ASC;
```

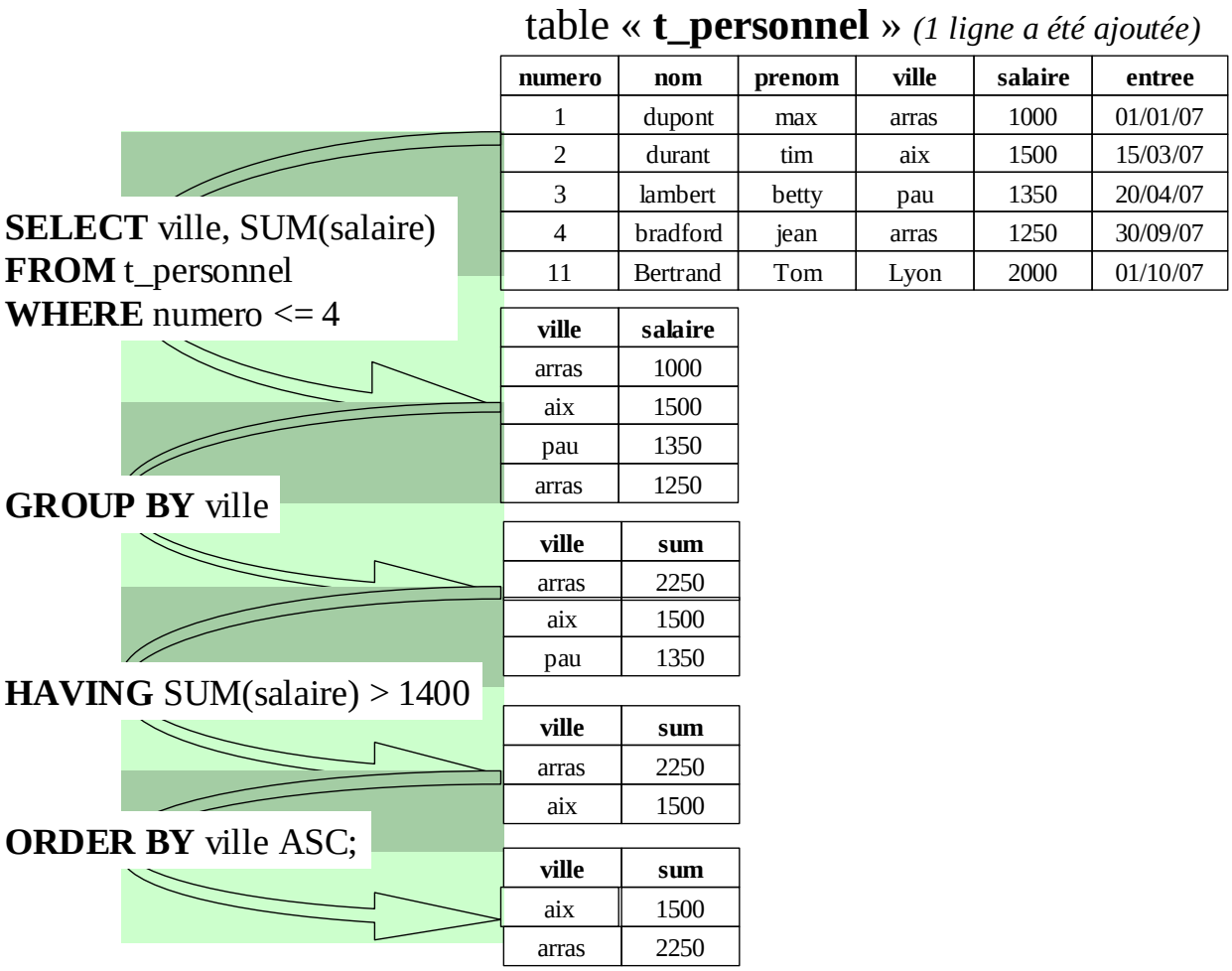


Figure 13 : succession des phases dans la production du résultat final

L'opération de projection est généralement réalisée en dernier.