

Partie III : Les Fonctions

Exercice 2.3.1 – Pourquoi lorsque l’on utilise la fonction « MAX » ou « MIN » les valeurs « NULL » sont-elles ignorées ?

Solution : Car elles n’ont pas d’impact sur la recherche de la valeur maximum ou minimum

Exercice 2.3.2 – Pourquoi le type des données n’a-t-il pas d’importance lorsque l’on utilise la fonction « COUNT » ?

Solution : La fonction « COUNT » compte le nombre de lignes qui correspondent au critère énoncé. Le type des données de la colonne choisie pour effectuer le « COUNT » n’a pas d’importance puisqu’il s’agit juste de tester si une valeur existe peu importe son type

Exercice 2.3.3 – La fonction « AVG » renvoie la moyenne de toutes les lignes résultantes d’une requête SELECT sur une colonne incluant toutes les valeurs « NULL ». (Vrai/Faux ?)

Solution : Faux, la fonction AVG ne tient pas compte des valeurs NULL

Exercice 2.3.4 – La fonction « SUM » est utilisée pour ajouter des totaux aux colonnes. (Vrai/Faux ?)

Solution : Faux, la fonction « SUM » a pour principe de renvoyer le résultat d’une somme de valeur. Lorsqu’elle est utilisée, seule une valeur peut-être proposée dans le résultat. La fonction somme ne peut donc ajouter une ligne contenant la somme en fin de colonne

Exercice 2.3.5 – La fonction « COUNT(*) » compte toutes les lignes d’une table. (Vrai/Faux ?)

Solution : Vrai, à la différence des autres versions de la fonction « COUNT », celle contenant l’étoile inclus dans les valeurs les lignes dont la valeur est « NULL »

Exercice 2.3.6 – Les requêtes suivantes sont-elles valides ?

```
1 SELECT COUNT *
2 FROM student;
3
4 SELECT COUNT(student_id), login
5 FROM student;
6
7 SELECT MIN(year_result), MAX(birth_date)
8 FROM student
9 WHERE year_result > 12;
```

Solution :

- Première requête, ligne 1 : il faut des parenthèses autour de l'étoile de la fonction « COUNT »
- Deuxième requête, ligne 4 : il est impossible de demander l'affichage d'une fonction d'agrégation (comme le « COUNT » dans ce cas-ci) en même temps que celui d'une colonne simple (« login », ici) sans avoir recours à un « GROUP BY » (vu dans la partie IV, plus bas)
- La dernière requête est correcte

```
1 SELECT COUNT(*)
2 FROM student;
3
4 SELECT COUNT(student_id)
5 FROM student;
6
7 SELECT MIN(year_result), MAX(birth_date)
8 FROM student
9 WHERE year result > 12;
```

Exercice 2.3.7 – Donner le résultat annuel moyen pour l'ensemble des étudiants

```
1 SELECT AVG(year_result) as 'Résultat annuel moyen'
2 FROM student
```

Exercice 2.3.8 – Donner le plus haut résultat annuel obtenu par un étudiant

```
1|SELECT MAX(year_result) as 'Plus haut résultat annuel'
2|FROM student
```

Exercice 2.3.9 – Donner la somme des résultats annuels

```
1|SELECT SUM(year_result) as 'Somme des résultats annuels'
2|FROM student
```

Exercice 2.3.10 – Donner le résultat annuel le plus faible

```
1|SELECT MIN(year_result) as 'Résultat annuel le plus faible'
2|FROM student
```

Exercice 2.3.11 – Donner le nombre de lignes qui composent la table « STUDENT »

```
1|SELECT COUNT(year_result) as 'Nombre de lignes'
2|FROM student
```

Exercice 2.3.12 – Donner la liste des étudiants (login et année de naissance) nés après 1970

```
1|SELECT login, DATEPART(yyyy, birth_date) as 'Année de naissance'
2|FROM student
3|WHERE YEAR(birth_date) > 1970
```

Exercice 2.3.13 – Donner le login et le nom de tous les étudiants qui ont un nom composé d'au moins 8 lettres

```
1 SELECT login, last_name
2 FROM student
3 WHERE LEN(last_name) >= 8
```

Exercice 2.3.14 – Donner la liste des étudiants ayant obtenu un résultat annuel supérieur ou égal à 16. La liste présente le nom de l'étudiant en majuscules (nommer la colonne « Nom de Famille ») et le prénom de l'étudiant dans l'ordre décroissant des résultats annuels obtenus

```
1 SELECT UPPER(last_name) AS 'Nom de famille', first_name, year_result
2 FROM student
3 WHERE year_result >= 16
4 ORDER BY year_result DESC
```

Exercice 2.3.15 – Donner un nouveau login à chacun des étudiants ayant obtenu un résultat annuel compris entre 6 et 10. Le login se compose des deux premières lettres du prénom de l'étudiant suivi par les quatre premières lettres de son nom le tout en minuscule. Le résultat reprend pour chaque étudiant, son nom, son prénom l'ancien et le nouveau login (colonne « Nouveau login »)

```
1 SELECT first_name, last_name, login,
2         LOWER(LEFT(first_name,2) + SUBSTRING(last_name,1,4)) AS 'Nouveau login'
3 FROM student
4 WHERE year_result BETWEEN 6 AND 10
```

Exercice 2.3.16 – Donner un nouveau login à chacun des étudiants ayant obtenu un résultat annuel égal à 10, 12 ou 14. Le login se compose des trois dernières lettres de son prénom suivi du chiffre obtenu en faisant la différence entre l'année en cours et l'année de leur naissance. Le résultat reprend pour chaque étudiant, son nom, son prénom l'ancien et le nouveau login (colonne « Nouveau login »)

```
1 SELECT first_name, last_name, login,  
2 RIGHT(first_name, 3)  
3 + CONVERT( VARCHAR, (datepart(yyyy, getDate()) - YEAR(birth_date)))  
4 AS 'Nouveau login'  
5 FROM student  
6 WHERE year result IN (10, 12, 14)
```

Exercice 2.3.17 – Donner la liste des étudiants (nom, login, résultat annuel) qui ont un nom commençant par « D », « M » ou « S ». La liste doit présenter les données dans l'ordre croissant des dates de naissance des étudiants

```
1 SELECT last_name, login, year_result  
2 FROM student  
3 WHERE UPPER(LEFT(last_name,1)) IN ('D', 'M', 'S')  
4 ORDER BY birth_date
```

Exercice 2.3.18 – Donner la liste des étudiants (nom, login, résultat annuel) qui ont obtenu un résultat impair supérieur à 10. La liste doit être triée du plus grand résultat au plus petit

```
1 SELECT last_name, login, year_result  
2 FROM student  
3 WHERE year_result % 2 = 1  
4 AND year_result > 10  
5 ORDER BY year result DESC
```

Exercice 2.3.19 – Donner le nombre d'étudiants qui ont au moins 7 lettres dans leur nom de famille

```
1 SELECT COUNT(last_name) AS 'Nbre de noms de plus de 7 lettres'  
2 FROM student  
3 WHERE LEN(last_name) >= 7
```

Exercice 2.3.20 – Pour chaque étudiant né avant 1955, donner le nom, le résultat annuel et le statut. Le statut prend la valeur « OK » si l'étudiant a obtenu au moins 12 comme résultat annuel et « KO » dans le cas contraire

```
1 SELECT last_name, year_result,  
2 CASE  
3     WHEN year_result >= 12 THEN 'OK'  
4     WHEN year_result < 12 THEN 'KO'  
5 END AS 'Statut'  
6 FROM student  
7 WHERE datepart(yy, birth_date) < 1955
```

Exercice 2.3.21 – Donner pour

chaque étudiant né entre 1955 et 1965 le nom, le résultat annuel et la catégorie à laquelle il appartient. La catégorie est fonction du résultat annuel obtenu : un résultat inférieur à 10 appartient à la catégorie « inférieure », un résultat égal à 10 appartient à la catégorie « neutre », un résultat autre appartient à la catégorie « supérieure »

```
1 SELECT last_name, year_result,  
2 CASE  
3     WHEN year_result < 10 THEN 'inférieure'  
4     WHEN year_result = 10 THEN 'neutre'  
5     ELSE 'supérieure'  
6 END AS 'Catégorie'  
7 FROM student  
8 WHERE YEAR(birth_date) BETWEEN 1955 and 1965
```

Exercice 2.3.22 – Donner pour chaque étudiant né entre 1975 et 1985, son nom, son résultat annuel et sa date de naissance sous la forme: jours en chiffre, mois en lettre et années en quatre chiffres (ex : 11 juin 2005)

```

1 SELECT last_name, year_result, CONVERT(VARCHAR, DAY(birth_date)) + ' ' +
2 CASE MONTH(birth_date)
3   WHEN 1 THEN 'janvier'
4   WHEN 2 THEN 'fevrier'
5   WHEN 3 THEN 'mars'
6   WHEN 4 THEN 'avril'
7   WHEN 5 THEN 'mai'
8   WHEN 6 THEN 'juin'
9   WHEN 7 THEN 'juillet'
10  WHEN 8 THEN 'août'
11  WHEN 9 THEN 'septembre'
12  WHEN 10 THEN 'octobre'
13  WHEN 11 THEN 'novembre'
14  WHEN 12 THEN 'décembre'
15 END
16 + ' ' + CONVERT(VARCHAR, YEAR(birth_date)) AS 'Literal_date'
17 FROM student
18 WHERE YEAR(birth_date) BETWEEN 1975 AND 1985

```

Exercice 2.3.23 – Donner pour chaque étudiant né en dehors des mois d’hiver et ayant obtenu un résultat inférieur à 7, son nom, le mois de sa naissance (en chiffre) son résultat annuel et son résultat annuel corrigé (« Nouveau résultat ») tel que si le résultat annuel est égal à 4, la valeur proposée est « NULL »

```

1 SELECT last_name, MONTH(birth_date) as 'Mois de naissance',
2 year_result, NULLIF(year_result, 4) as 'Nouveau résultat'
3 FROM student
4 WHERE DATEPART(MONTH, birth_date) NOT IN (12, 1, 2, 3)
5 AND year_result < 7

```

Partie IV : GROUP BY ... HAVING

Exercice 2.4.1 – L'utilisation de « GROUP BY » peut être considérée comme une forme de boucle dans une requête SQL ? (Vrai/Faux)

Solution : Si l'on considère que le système applique plusieurs fois la même fonction agrégative à des ensembles de données différents, le « GROUP BY » s'apparente en quelques sortes à une boucle

Exercice 2.4.2 – La répartition en groupe se fait avant de prendre en compte les restrictions imposées par un « WHERE » ? (Vrai/Faux)

Solution : Faux. Le système passe dans le « WHERE » avant de s'attaquer au « GROUP BY », ce qui lui permet d'éliminer un certains nombres de lignes et d'alléger le traitement

Exercice 2.4.3 – Un « GROUP BY » doit impérativement porter sur une colonne ne possédant pas d'alias ?

Solution : Vrai. Comme le système n'est pas encore passé dans le « SELECT » au moment de traiter le « GROUP BY », il ne connaît pas encore les alias que l'on a éventuellement créés et ne peut donc pas les utiliser

Exercice 2.4.4 – L'utilisation d'un « GROUP BY » a pour effet de trier les résultats dans l'ordre croissant de la colonne incluse dans le « GROUP BY » ? (Vrai/Faux)

Solution : Cela s'avère vrai sur la plupart des systèmes

Exercice 2.4.5 – La colonne sur laquelle porte le « GROUP BY » doit impérativement être présente dans la clause « SELECT » ? (Vrai/Faux)

Solution : Faux. On peut tout à fait n'afficher que le résultat du « GROUP BY » sans le faire correspondre à son groupe, à nous de savoir à quoi correspond l'affichage. C'est d'ailleurs ce que nous devons faire dans les requêtes imbriquées de la partie VII

Exercice 2.4.6 – Les requêtes suivantes sont-elles valides ?

```
1 SELECT section_id, count(last_name)
2 FROM student
3 GROUP BY last_name;
4
5 SELECT section_id, AVG(year_result)
6 FROM student
7 WHERE AVG(year_result) > 50
8 GROUP BY section_id;
9
10 SELECT section_id, AVG(year_result)
11 FROM student
12 WHERE year_result > 10
13 GROUP BY section_id;
```

Solution :

- Première requête, ligne 3 : **toutes les colonnes du « SELECT » qui ne font pas partie de la fonction d'agrégation DOIVENT se retrouver dans le « GROUP BY »**. Il faut donc par exemple remplacer « last_name » par « section_id »
- Deuxième requête, ligne 7 : une condition posée sur un sous-groupe de données doit se trouver dans le « HAVING » du « GROUP BY ». Le « WHERE » sert à éliminer des lignes, indépendamment des fonctions d'agrégation.
- La dernière requête est correcte : le « WHERE » limite les lignes traitées au départ et le « GROUP BY » regroupe les données auxquelles il faut appliquer la fonction d'agrégation

```
1 SELECT section_id, count(last_name)
2 FROM student
3 GROUP BY section_id;
4
5 SELECT section_id, AVG(year_result)
6 FROM student
7 GROUP BY section_id
8 HAVING AVG(year_result) > 10;
9
10 SELECT section_id, AVG(year_result)
11 FROM student
12 WHERE year_result > 10
13 GROUP BY section id;
```

Exercice 2.4.7 – Donner pour chaque section, le résultat maximum (dans une colonne appelée « Résultat maximum ») obtenu par les étudiants

```
1 SELECT section_id, MAX(year_result) as 'Résultat maximum'
2 FROM student
3 GROUP BY section_id
```

Exercice 2.4.8 – Donner pour toutes les sections commençant par 10, le résultat annuel moyen PRÉCIS (dans une colonne appelée « Moyenne ») obtenu par les étudiants

```
1 SELECT section_id, AVG(CONVERT(FLOAT, year_result)) as 'Moyenne'
2 FROM student
3 WHERE section_id LIKE '10__'
4 GROUP BY section_id
```

Exercice 2.4.9 – Donner le résultat moyen (dans une colonne appelée « Moyenne ») et le mois en chiffre (dans une colonne appelée « Mois de naissance ») pour les étudiants nés le même mois entre 1970 et 1985

```
1 SELECT MONTH(birth_date) as 'Mois de naissance',
2        AVG(year_result) as 'Moyenne'
3 FROM student
4 WHERE DATEPART(YYYY, birth_date) BETWEEN 1970 AND 1985
5 GROUP BY DATEPART(MONTH, birth_date)
```

Exercice 2.4.10 – Donner pour toutes les sections qui comptent plus de 3 étudiants, la moyenne PRÉCISE des résultats annuels (dans une colonne appelée « Moyenne »)

```

1 SELECT section_id, AVG(CONVERT(FLOAT,year_result)) as 'Moyenne'
2 FROM student
3 GROUP BY section_id
4 HAVING COUNT(last name) > 3

```

Exercice 2.4.11 – Donner le résultat maximum obtenu par les étudiants appartenant aux sections dont le résultat moyen est supérieur à 8

```

1 SELECT section_id, AVG(year_result) as 'Moyenne',
2         MAX(year_result) as 'Résultat maxium'
3 FROM student
4 GROUP BY section_id
5 HAVING AVG(year result) > 8

```

Partie V : CUBE et ROLLUP

Exercice 2.5.1 – L'utilisation de « ROLLUP » crée des groupes de données en se déplaçant dans une seule direction, partant de la gauche vers la droite par rapport aux colonnes sélectionnées ? (Vrai/Faux)

Solution : Vrai. Au niveau de l'affichage, le système suivra toujours l'ordre demandé par le select. Cependant, la fonction de regroupement est appliquée en remontant les colonnes de la droite vers la gauche.

Exercice 2.5.2 – Le résultat produit par un « ROLLUP » présente les résultats du plus agrégé au moins agrégé ? (Vrai/Faux)

Solution : Faux, il n'y a pas de tri sur la façon dont les données sont regroupées

Exercice 2.5.3 – L'opérateur « CUBE » permet de produire moins de sous-totaux qu'avec l'opérateur « ROLLUP » ? (Vrai/Faux)

Solution : Faux, c'est l'inverse puisque le « CUBE » rajoute des agrégations supplémentaires

Exercice 2.5.4 – Avec l'opérateur « CUBE », le nombre de groupes dans le résultat est indépendant du nombre de colonnes sélectionnées dans le « GROUP BY » ? (Vrai/Faux)

Solution : Faux, le « CUBE » est directement lié au « GROUP BY »

Exercice 2.5.5 – L'opérateur « CUBE » ne peut pas être appliqué à la fonction d'agrégation « SUM » ? (Vrai/Faux)

Solution : Faux, le « CUBE » peut être utilisé avec toutes les fonctions d'agrégation

Exercice 2.5.6 – Donner la moyenne exacte des résultats obtenus par les étudiants par section et par cours, ainsi que la moyenne par section uniquement et enfin, la moyenne générale. La liste ainsi produite reprend l'id de section, de cours le résultat moyen (dans une colonne appelée « Moyenne »). Se baser uniquement sur les sections 1010 et 1320

```
1 SELECT section_id, course_id,  
2         AVG(CONVERT(FLOAT, year_result)) as 'Moyenne'  
3 FROM student  
4 WHERE section_id IN (1010, 1320)  
5 GROUP BY section id, course id WITH ROLLUP
```

Exercice 2.5.7 – Donner la moyenne exacte des résultats obtenus par les étudiants par cours et par section, ainsi que la moyenne par cours uniquement, puis par section uniquement et enfin, la moyenne générale. La liste ainsi produite reprend l'id de section, de cours le résultat moyen (dans une colonne appelée « Moyenne »). Se baser uniquement sur les sections 1010 et 1320

```
1 SELECT course_id, section_id,  
2     AVG(CONVERT(FLOAT, year_result)) as 'Moyenne'  
3 FROM student  
4 WHERE section_id IN (1010, 1320)  
5 GROUP BY course id, section id WITH CUBE
```