MODULE 2 :

DRL (Data Retrieval Language - Langage d'extraction de données)

Partie I : SELECT … FROM …

Exercice 2.1.1 – Les requêtes suivantes fonctionnent-elles sous SQL-Server ? Si non, comment les corriger ?
N'hésitez pas à tester vos requêtes directement sous Management Studio !

```
1  SELECT last_name, first_name AS F name
2  FROM student;
3
4  SELECT last_name lname, first_name AS fname
5  FROM student;
6
7  SELECT last_name|| _ ||first_name AS name
8  FROM student;
9
10 SELECT last_name+first_name AS name, Year_result x 10 result,
11 FROM student;
```

Solution :

- Première requête, ligne1 : il faut rajouter des guillemets (simples ou doubles) autour de « F name » ou bien remplacer l'espace par un « underscore »
- La deuxième requête fonctionne très bien car le « AS » est facultatif
- Troisième requête, ligne 7 : la concaténation se fait avec le caractère « + », sous SQL-Server, pas les « || » qu'il faut donc remplacer ; ensuite, le caractère « _ » (underscore) doit être entre guillemets simples
- Quatrième requête, ligne 10 : la multiplication se symbolise par l'étoile « * » et non le « x » qu'il faut donc remplacer ; la virgule en fin de ligne créera une erreur, elle est à supprimer

```
1  SELECT last_name, first_name AS 'F name'
2  FROM student;
3
4  SELECT last_name lname, first_name AS fname
5  FROM student;
6
7  SELECT last_name + '_' + first_name AS name
8  FROM student;
9
10 SELECT last_name+first_name AS name, Year_result * 10 result
11 FROM student;
```

Exercice 2.1.2 – **Ecrire une requête pour présenter, pour chaque étudiant, le nom de l'étudiant, la date de naissance, le login et le résultat pour l'année de l'ensemble des étudiants.**

```
SELECT last_name, birth_date, login, year_result
FROM student
```

Exercice 2.1.3 – **Ecrire une requête pour présenter, pour chaque étudiant, son nom complet (nom et prénom séparés par un espace), son id et sa date de naissance.**

```
SELECT last_name + ' ' + first_name as 'Nom complet', student_id, birth_date
FROM student
```

Exercice 2.1.4 – **Ecrire une requête pour présenter, pour chaque étudiant, dans une seule colonne (nommée « Info Étudiant ») l'ensemble des données relatives à un étudiant séparées par le symbole « | ». Il est nécessaire d'avoir recours à la fonction de conversion CONVERT(type, champs).**

```
SELECT CONVERT(VARCHAR, student_id) + '|' + last_name + '|'
       + first_name + '|' + CONVERT(VARCHAR, birth_date) + '|'
       + login +'|' + CONVERT(VARCHAR, section_id) +'|'
       + CONVERT(VARCHAR, year_result) as 'Info Étudiant'
FROM student
```

Partie II : SELECT … FROM … WHERE … ORDER BY

Exercice 2.2.1 – **Ecrire une requête pour présenter le login et le résultat de tous les étudiants ayant obtenu un résultat annuel supérieur à 16**

```
SELECT login, year_result
 FROM student
WHERE year result > 16
```

Exercice 2.2.2 – **Ecrire une requête pour présenter le nom et l'id de section des étudiants dont le prénom est Georges**

```
SELECT last_name, section_id
 FROM student
WHERE first name like 'Georges'
```

Exercice 2.2.3 – **Ecrire une requête pour présenter le nom et le résultat annuel de tous les étudiants ayant obtenu un résultat annuel compris entre 12 et 16**

```
SELECT last_name, year_result
 FROM student
WHERE year result BETWEEN 12 AND 16
```

Exercice 2.2.4 – **Ecrire une requête pour présenter le nom, l'id de section et le résultat annuel de tous les étudiants qui ne font pas partie des sections 1010, 1020 et 1110**

```
SELECT last_name, section_id, year_result
 FROM student
WHERE section id NOT IN (1010, 1020, 1110)
```

Exercice 2.2.5 – **Ecrire une requête pour présenter le nom et l'id de section de tous les étudiants qui ont un nom de famille qui termine par « r »**

```
SELECT last_name, section_id
 FROM student
WHERE last name LIKE '%r'
```

Exercice 2.2.6 – **Ecrire une requête pour présenter le nom et le résultat annuel de tous les étudiants qui ont un nom de famille pour lequel la troisième lettre est un « n » et qui ont obtenu un résultat annuel supérieur à 10**

```
SELECT last_name, year_result
 FROM student
 WHERE last_name LIKE '__n%'
 AND year_result > 10
```

Exercice 2.2.7 – **Ecrire une requête pour présenter le nom et le résultat annuel classé par résultats annuels décroissants de tous les étudiants qui ont obtenu un résultat annuel inférieur ou égal à 3**

```
SELECT last_name, year_result
 FROM student
 WHERE year_result <= 3
 ORDER BY year result DESC
```

Exercice 2.2.8 – **Ecrire une requête pour présenter le nom complet (nom et prénom séparés par un espace) et le résultat annuel classé par nom croissant sur le nom de tous les étudiants appartenant à la section 1010**

```
SELECT last_name + ' ' + first_name as 'Nom complet', year_result
 FROM student
 WHERE section_id = 1010
 ORDER BY last_name ASC
```

Exercice 2.2.9 – **Ecrire une requête pour présenter le nom, l'id de section et le résultat annuel classé par ordre croissant sur la section de tous les étudiants appartenant aux sections 1010 et 1020 ayant un résultat annuel qui n'est pas compris entre 12 et 18**

```
SELECT last_name, section_id, year_result
 FROM student
 WHERE section_id IN (1010, 1020)
       AND year_result NOT BETWEEN 12 AND 18
 ORDER BY section_id
```

Exercice 2.2.10 – **Ecrire une requête pour présenter le nom, l'id de section et le résultat annuel sur 100 (nommer la colonne « Résultat sur 100 ») classé par ordre décroissant du résultat de tous les étudiants appartenant aux sections commençant par 13 et ayant un résultat annuel sur 100 inférieur ou égal à 60**

```
1 SELECT last_name, section_id, (year_result*5) as 'Résultat sur 100'
2  FROM student
3  WHERE section_id like '13__' AND year_result*5 <= 60
4 ORDER BY year_result DESC
```

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**

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

Exercice 2.3.9 – **Donner la somme des résultats annuels**

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

Exercice 2.3.10 – **Donner le résultat annuel le plus faible**

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

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

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

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

```
SELECT login, DATEPART(yyyy, birth_date) as 'Année de naissance'
 FROM student
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**

```
SELECT login, last_name
 FROM student
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**

```
SELECT UPPER(last_name) AS 'Nom de famille', first_name, year_result
 FROM student
 WHERE year_result >= 16
 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 »)**

```
SELECT first_name, last_name, login,
       LOWER(LEFT(first_name,2) + SUBSTRING(last_name,1,4)) AS 'Nouveau login'
 FROM student
 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 »)**

```
SELECT first_name, last_name, login,
 RIGHT(first_name, 3)
 + CONVERT( VARCHAR, (datepart(yyyy, getDate()) - YEAR(birth_date)))
 AS 'Nouveau login'
 FROM student
 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 à 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(yyyy, 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 »**

```
SELECT last_name, year_result,
 CASE
   WHEN year_result < 10 THEN 'inferieure'
   WHEN year_result = 10 THEN 'neutre'
   ELSE 'superieure'
 END AS 'Catégorie'
 FROM student
 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)**

```
SELECT last_name, year_result, CONVERT(VARCHAR,DAY(birth_Date)) + ' ' +
 CASE MONTH(birth_date)
   WHEN 1 THEN 'janvier'
   WHEN 2 THEN 'fevrier'
   WHEN 3 THEN 'mars'
   WHEN 4 THEN 'avril'
   WHEN 5 THEN 'mai'
   WHEN 6 THEN 'juin'
   WHEN 7 THEN 'juillet'
   WHEN 8 THEN 'août'
   WHEN 9 THEN 'septembre'
   WHEN 10 THEN 'octobre'
   WHEN 11 THEN 'novembre'
   WHEN 12 THEN 'décembre'
 END
 + ' ' + CONVERT(VARCHAR,YEAR(birth_date)) AS 'Literal_date'
 FROM student
 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, le valeur proposée est « NULL »**

```
SELECT last_name, MONTH(birth_date) as 'Mois de naissance',
year_result, NULLIF(year_result, 4) as 'Nouveau résultat'
FROM student
WHERE DATEPART(MONTH, birth_date) NOT IN (12, 1, 2, 3)
      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 non alliacée ?

Solution : Vrai. Comme le système n'est pas encore passé dans le « SELECT » au moment de traiter le « GROUP BY », il ne connait pas encore les allias 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 devrons 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**

```
SELECT section_id, MAX(year_result) as 'Résultat maximum'
 FROM student
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**

```
SELECT section_id, AVG(CONVERT(FLOAT,year_result)) as 'Moyenne'
 FROM student
 WHERE section_id LIKE '10__'
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**

```
SELECT MONTH(birth_date) as 'Mois de naissance',
       AVG(year_result) as 'Moyenne'
 FROM student
 WHERE DATEPART(YYYY, birth_date) BETWEEN 1970 AND 1985
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 »)**

```
SELECT section_id, AVG(CONVERT(FLOAT,year_result)) as 'Moyenne'
 FROM student
 GROUP BY section_id
HAVING COUNT(last_name) > 3
```

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

```
SELECT section_id, AVG(year_result) as 'Moyenne',
       MAX(year_result) as 'Résultat maxium'
FROM student
GROUP BY section_id
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**

```
SELECT section_id, course_id,
       AVG(CONVERT(FLOAT,year_result)) as 'Moyenne'
FROM student
WHERE section_id IN (1010, 1320)
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**

```
SELECT course_id, section_id,
       AVG(CONVERT(FLOAT,year_result)) as 'Moyenne'
FROM student
WHERE section_id IN (1010, 1320)
GROUP BY course_id, section_id WITH CUBE
```

Partie VI : Jointures

Exercice 2.6.1 – **Donner pour chaque cours le nom du professeur responsable ainsi que la section dont le professeur fait partie**

```
1 SELECT course_name, section_name, professor_name
2 FROM professor P, course C, section S
3 WHERE S.section_id = P.section_id
4 AND P.professor_id = C.professor_id
```

```
1 SELECT course_name, section_name, professor_name
2 FROM professor P JOIN course C
3      ON P.professor_id = C.professor_id
4      JOIN section S ON S.section_id = P.section_id
```

Exercice 2.6.2 – **Donner pour chaque section, l'id, le nom et le nom de son délégué. Classer les sections dans l'ordre inverse des id de section. Un délégué est un étudiant de la table « STUDENT »**

```
1 SELECT S.section_id, S.section_name, St.last_name
2 FROM section S, student St
3 WHERE S.delegate_id = St.student_id
4 ORDER BY S.section_id DESC
```

```
1 SELECT S.section_id, S.section_name, St.last_name
2 FROM section S JOIN student St
3      ON S.delegate_id = St.student_id
4 ORDER BY S.section_id DESC
```

Exercice 2.6.3 – **Donner pour chaque section, le nom des professeurs qui en sont membre**

```
SELECT S.section_id, S.section_name, P.professor_name
FROM section S LEFT JOIN professor P
     ON S.section_id = P.section_id
ORDER BY S.section_id DESC
```

Exercice 2.6.4 – **Même objectif que la question 3 mais seuls les sections comportant au moins un professeur doivent être reprises**

```
SELECT S.section_id, S.section_name, P.professor_name
FROM section S JOIN professor P
     ON S.section_id = P.section_id
ORDER BY S.section_id DESC
```

Exercice 2.6.5 – **Donner à chaque étudiant ayant obtenu un résultat annuel supérieur ou égal à 12 son grade en fonction de son résultat annuel et sur base de la table grade. La liste doit être classée dans l'ordre alphabétique des grades attribués**

```
SELECT S.last_name, S.year_result, G.Grade
FROM Grade AS G, student AS S
WHERE (S.year_result BETWEEN G.Lower_bound AND G.Upper_bound)
      AND (S.year_result >= 12)
ORDER BY G.Grade
```

Exercice 2.6.6 – **Donner la liste des professeurs et la section à laquelle ils se rapportent ainsi que le(s) cour(s) (nom du cours et crédits) dont le professeur est responsable. La liste est triée par ordre décroissant des crédits attribués à un cours**

```
SELECT P.professor_name, section_name, C.course_name, C.course_ects
FROM professor AS P LEFT JOIN section AS S
     ON P.section_id = S.section_id,
     professor AS P1 LEFT JOIN course AS C
     ON P1.professor_id = C.professor_id
WHERE P1.professor_id = P.professor_id
ORDER BY C.course_ects DESC
```

```
SELECT P.professor_name, section_name, C.course_name, C.course_ects
FROM section AS S RIGHT JOIN professor AS P
     ON P.section_id = S.section_id
     LEFT JOIN course AS C
     ON P.professor_id = C.professor_id
ORDER BY C.course_ects DESC
```

Exercice 2.6.7 – **Donner pour chaque professeur son id et le total des crédits ECTS (« ECTS_TOT ») qui lui sont attribués. La liste proposée est triée par ordre décroissant de la somme des crédits alloués**

```
SELECT P.professor_id, sum(C.course_ects) AS ECTS_TOT
FROM professor AS P LEFT JOIN course AS C
     ON p.professor_id=c.professor_id
GROUP BY P.professor_id
ORDER BY sum(C.course_ects) DESC
```

Exercice 2.6.8 – **Donner la liste (nom et prénom) de l'ensemble des professeurs et des étudiants dont le nom est composé de plus de 8 lettres. Ajouter une colonne pour préciser la catégorie (S pour « STUDENT », P pour « PROFESSOR ») à laquelle appartient l'individu**

```
SELECT first_name, last_name, 'S' as "Catégorie"
FROM student
WHERE LEN(last_name) > 8

UNION

SELECT professor_surname, professor_name, 'P'
FROM professor
WHERE LEN(professor_name) > 8
```

Exercice 2.6.9 – **Donner l'id de chacune des sections qui n'ont pas de professeur attitré**

```
SELECT section_id FROM section

EXCEPT

SELECT section_id FROM professor
```

Partie VII : Requêtes imbriquées

Exercice 2.7.1 – **Donner la liste des étudiants (nom et prénom) qui font partie de la même section que mademoiselle « Roberts ». La liste doit être classée par ordre alphabétique sur le nom et mademoiselle « Roberts » ne doit pas apparaitre dans la liste**

```
SELECT last_name, first_name, section_id
FROM student
WHERE section_id = ( SELECT section_id
                     FROM student
                     WHERE last_name='Roberts' )
      AND last_name <> 'Roberts'
ORDER BY last_name
```

Exercice 2.7.2 – **Donner la liste des étudiants (nom, prénom et résultat) de l'ensemble des étudiants ayant obtenu un résultat annuel supérieur au double du résultat moyen pour l'ensemble des étudiants**

```
SELECT last_name, first_name, year_result
FROM student
WHERE year_result > 2 * ( SELECT AVG(year_result)
                          FROM student )
```

Exercice 2.7.3 – **Donner la liste de toutes les sections qui n'ont pas de professeur**

```
SELECT section_id, section_name
FROM section
WHERE section_id NOT IN ( SELECT DISTINCT section_id
                          FROM professor )
```

Exercice 2.7.4 – **Donner la liste des étudiants qui ont comme mois de naissance le mois correspondant à la date d'engagement du professeur « Giot ». Classer les étudiants par ordre de résultat annuel décroissant**

```
SELECT last_name, first_name,
       CONVERT(VARCHAR,birth_date,101) AS "Date de Naissance",
       year_result
FROM student
WHERE MONTH(birth_date) = ( SELECT DATEPART(MONTH,professor_hire_date)
                            FROM professor
                            WHERE professor_name = 'Giot' )
ORDER BY year_result DESC
```

Exercice 2.7.5 – **Donner la liste des étudiants qui ont obtenu le grade « TB » pour leur résultat annuel**

```
SELECT last_name, first_name, year_result
FROM student
WHERE year_result BETWEEN ( SELECT Lower_bound
                            FROM Grade
                            WHERE grade like 'TB' )
                  AND ( SELECT Upper_bound
                        FROM Grade
                        WHERE grade like 'TB' )
```

Exercice 2.7.6 – **Donner la liste des étudiants qui appartienne à la section pour laquelle Mademoiselle « Marceau » est déléguée**

```
SELECT last_name, first_name, section_id
FROM student
WHERE section_id = ( SELECT section_id
                     FROM section
                     WHERE delegate_id = ( SELECT student_id
                                           FROM student
                                           WHERE last_name like 'Marceau' ) )
```

Exercice 2.7.7 – **Donner la liste des sections qui se composent de plus de quatre étudiants**

```sql
SELECT section_id, section_name
FROM section
WHERE section_id IN ( SELECT section_id
                      FROM student
                      GROUP BY section_id
                      HAVING COUNT(section_id) > 4 )
```

Exercice 2.7.8 – **Donner la liste des étudiants premiers de leur section en terme de résultat annuel et qui n'appartiennent pas aux sections dont le résultat moyen est inférieure à 10**

```sql
-- Moyenne par section
SELECT AVG(year_result)
FROM student
GROUP BY section_id

-- Sections dont le résultat moyen est inférieure à 10
SELECT section_id
FROM student
GROUP BY section_id
HAVING AVG(year_result) < 10

-- Étudiants qui ne sont pas dans les section
-- dont le résultat moyen est inférieure à 10
SELECT *
FROM student
WHERE section_id NOT IN (
    SELECT section_id
    FROM student
    GROUP BY section_id
    HAVING AVG(year_result) < 10
)

-- Meilleur résultat par section
SELECT section_id, MAX(year_result)
FROM student
GROUP BY section_id
```

```sql
-- Info du meilleur étudiant par section
SELECT last_name, first_name, section_id
FROM student s
WHERE year_result = (
    SELECT MAX(year_result)
    FROM student s1
    WHERE s.section_id = s1.section_id
    GROUP BY section_id
)

```

```sql
-- Combiné des deux parties - version finale
SELECT last_name, first_name, section_id
FROM student s
WHERE section_id NOT IN (
    SELECT section_id
    FROM student
    GROUP BY section_id
    HAVING AVG(year_result) < 10
) AND year_result = (
    SELECT MAX(year_result)
    FROM student s1
    WHERE s.section_id = s1.section_id
    GROUP BY section_id
)
```

Solution utilisant une jointure ainsi qu'une requête imbriquée dans la clause « FROM »

```sql
select last_name,first_name,S1.section_id
from student as S1 inner join (select section_id, AVG(year_result) as average
                                 from student
                               group by section_id
                                 having AVG(year_result) >= 10)
            as S2 on S1.section_id = S2.section_id
where year_result = (select MAX(year_result)
                       from student
                      where section_id = S1.section_id)
```

Exercice 2.7.9 – **Donner la section qui possède la moyenne la plus élevée. Le résultat présente le numéro de section ainsi que sa moyenne**

```sql
-- 1. Trouver la moyenne pour chaque section
SELECT section_id, AVG(year_result) as 'Moyenne'
FROM student
GROUP BY section_id
HAVING AVG(year_result) = 17


-- 2. Trouver la moyenne la plus élevée parmi les moyennes de chaque section
SELECT MAX(Maximum)
FROM (SELECT section_id, AVG(year_result) as 'Maximum'
FROM student
GROUP BY section_id) AS Nouvelle_Table


```

```sql
-- 3. Trouver la section qui possède la moyennes la plus élevée
-- 4. Indiquer le numéro et la moyenne de la section qui possède la moyennes la plus élevée

SELECT section_id, AVG(year_result) as 'Moyenne'
FROM student
GROUP BY section_id
HAVING AVG(year_result) = (
SELECT MAX(Maximum)
FROM (
SELECT section_id, AVG(year_result) as 'Maximum'
FROM student
GROUP BY section_id) AS Nouvelle_Table
)


```

```sql
-- Méthode Avec WITH
WITH moy_par_sect (section_id, average) AS (
    SELECT section_id, AVG(year_result)
    FROM student
    GROUP BY section_id
)
SELECT section_id, average 'AVG'
FROM moy_par_sect mps
WHERE mps.average = (
    SELECT MAX(average)
    FROM moy_par_sect
);


-- Même résultat avec TOP
SELECT TOP(1) section_id, AVG(year_result)
FROM student
GROUP BY section_id
ORDER BY AVG(year_result) DESC
```