3
AS Montferrand saison 2016/2017

1. Une base de données est composée de **relations** : en pratique, une relation est une table, constituée d'un *en-tête* où figure la liste des **attributs** et d'un *corps*, qui est l'ensemble des **enregistrements** (les lignes de la table).
En pratique, un attribut est une grandeur identifiée dans l'en-tête de la table, dont les valeurs sont enregistrées dans les lignes du corps de la table. On peut donc imaginer un attribut comme une *colonne* d'une table.

2. L'ensemble des valeurs possibles d'un attribut (entiers, flottants, chaînes de caractères...) est le **domaine** de l'attribut.

3. Une **clé primaire** est une donnée qui permet d'identifier *de manière unique* un enregistrement d'une table. Une clé primaire peut être la valeur d'un attribut créé spécialement à cet effet ou se composer de la liste des valeurs de plusieurs attributs.

I
La base de données ASMontferrand.sqlite

4. La base de données `ASMontferrand.sqlite` a été établie à partir de données collectées par Thierry Wagner pour un site de supporteurs (`www.cybervulcans.net/modules/bd/`) et contient les données relatives à la saison 2016/2017, au terme de laquelle l'AS Montferrand a gagné le titre de champion de France de rugby.
La base de données `ASMontferrand.sqlite` est constituée de six tables.

4.1 La table `Effectif` regroupe les données relatives aux différents joueurs.

Attributs de la table `Effectif`

`idJoueur`	`Integer`	Clé primaire
`Nationalité`	`Varchar`	Nationalité du joueur (code sur deux lettres)
`Joueur`	`Varchar`	Identité du joueur (NOM Prénom)
`Né le`	`Date`	Date de naissance (JJ/MM/AAAA)
`International`	`Boolean`	1 pour les joueurs qui ont déjà évolué dans leur équipe nationale
`Taille`	`Integer`	Taille du joueur en centimètres
`Poids`	`Integer`	Masse du joueur en kilogrammes
`Temps de jeu`	`Integer`	Minutes passées sur le terrain pendant la saison

4.2 La table `Matchs` contient les informations relatives aux matchs officiels joués, tant en championnat de France (*Top14*) qu'en coupe d'Europe (*Champions Cup*).

Attributs de la table `Matchs`		
`rowid`	`Integer`	Clé primaire
`Date`	`Datetime`	Date du match (AAAA/MM/JJ)
`Adversaire`	`Varchar`	Nom du club rencontré par l'ASM
`Terrain`	`Integer`	1 pour match à domicile, 2 pour match à l'extérieur 0 pour terrain neutre
`Points Montferrand`	`Integer`	Score de l'ASM
`Points adversaire`	`Integer`	Score de l'équipe adverse
`Essais`	`Integer`	Nombre d'essais marqués par l'ASM
`Transformations`	`Integer`	Nombre d'essais transformés par l'ASM
`Pénalités`	`Integer`	Nombre de pénalités transformées par l'ASM
`Drops`	`Integer`	Nombre de drop passés par l'ASM
`Cartons jaunes`	`Integer`	Nombre d'exclusions temporaires
`Cartons rouges`	`Integer`	Nombre d'exclusions définitives
`Compétition`	`Varchar`	Type de compétition : championnat de France ou coupe d'Europe
`Niveau`	`Varchar`	Journée de championnat (Jxx) ou phase de coupe d'Europe

La phase régulière du championnat et la phase de poules de coupe d'Europe sont disputées par matchs aller et retour. En revanche, les demi-finales et la finale sont disputées sur terrain neutre, aussi bien pour le championnat de France que pour la coupe d'Europe.

4.3 Les manières de marquer des points

Un joueur marque un **essai** (5 points) en déposant le ballon dans la zone d'en-but adverse. À la suite d'un essai, l'équipe peut marquer 2 points supplémentaires en faisant passer le ballon entre les poteaux adverses par un coup de pied placé.

Lorsque l'équipe qui défend commet de nombreuses irrégularités à proximité de sa zone d'en-but, l'arbitre peut attribuer un essai à l'équipe qui attaque, dit **essai de pénalité**, qui n'est attribué à aucun joueur.

À la suite d'une faute adverse, l'arbitre peut attribuer une **pénalité**. On obtient 3 points en faisant passer le ballon entre les poteaux adverses par un coup de pied placé.

Dans le cours du jeu, si un joueur arrive à faire passer le ballon entre les poteaux adverses par un coup de pied tombé (c'est-à-dire en laissant tomber le ballon avant de taper dedans), son équipe marque les 3 points attribués pour un **drop goal**.

La table `ValeursPoints` indique les différentes manières de marquer des points au rugby et le nombre de points qui correspond à chaque manière.

Table `ValeursPoints`		
`idPoints (Char)`	`NomPoints (Varchar)`	`Valeur (Integer)`
0	Essai	5
1	Transformation	2
2	Pénalité	3
3	Drop goal	3

4.4 La table `Points` détaille les points marqués par l'AS Montferrand pendant cette saison.

Attributs de la table `Points`		
`idMatch`	`Integer`	Identifiant du match (table `Matchs`)
`Points`	`Char`	Identifiant du type de points marqués (table `ValeursPoints`)
`idJoueur`	`Integer`	Identifiant du joueur ayant marqué (table `Effectif`) 0 en cas d'essai de pénalisation
`Minute de jeu`	`Integer`	Instant du match auquel les points ont été marqués

4.5 La table `Cartons` regroupe les sanctions graves prises contre des joueurs de l'ASM. Un **carton jaune** exclut le joueur fautif du terrain pendant 10 minutes. Un **carton rouge** exclut le joueur pour le reste du match.

Attributs de la table `Cartons`		
`idMatch`	`Integer`	Identifiant du match (table `Matchs`)
`Carton`	`Char`	J pour carton jaune R pour carton rouge
`idJoueur`	`Integer`	Identifiant du joueur fautif (table `Effectif`)
`Minute de jeu`	`Integer`	Instant du match auquel le joueur a été sanctionné

4.6 La table `Compositions` donne la liste des 23 joueurs qui figurent sur chaque feuille de match. Les joueurs **titulaires** (ceux qui sont sur le terrain dès le début de la rencontre) portent un numéro compris entre 1 et 15 en fonction du poste qu'ils occupent.
Les **remplaçants** éventuels portent un numéro compris entre 16 et 23.

Attributs de la table `Compositions`		
`idMatch`	`Integer`	Identifiant du match (table `Matchs`)
`Poste`	`Integer`	Numéro du maillot
`idJoueur`	`Integer`	Identifiant du joueur (table `Effectif`)

Les **avants** sont les joueurs qui portent les numéros 1 à 8.
- La **première ligne** est constituée d'un **pilier gauche** (1), d'un **talonneur** (2) et d'un **pilier droit** (3).
- La **deuxième ligne** est constituée par les joueurs qui portent les numéros 4 et 5.
- La **troisième ligne** est constituée par les deux **troisième ligne aile** (6 et 7) et le **troisième ligne centre** (8).

La **charnière** est constituée par le **demi de mêlée** (numéro 9) et le **demi d'ouverture** (numéro 10).
Les **arrières** portent les numéros 11 à 15 : on distingue les **trois-quarts centre** (12 et 13), les **trois-quarts aile** (11 et 14) et l'**arrière** (15).

II

Requêtes simples

5. Sélection

La **sélection**, ou **restriction**, consiste à extraire d'une relation `table_A` la totalité des enregistrements qui vérifient un certain critère.
Pour éviter d'avoir à écrire la liste des tous les attributs, on utilise le caractère spécial $*$.

```
SELECT * FROM table_A
WHERE critère
```

6. Projection

Une **projection** consiste à extraire une partie des attributs d'une relation.
La projection plus simple consiste à dresser la table des valeurs prises par un attribut `champ_A` de la table `table_A`.

```
SELECT champ_A FROM table_A
```

Pour dresser la table des valeurs prises par plusieurs attributs d'une même table, on écrit les noms des différents attributs en les séparant par des virgules.

```
SELECT champ_A, champ_B, champ_C
FROM table_A
```

7. On peut filtrer la liste des enregistrements à traiter en imposant une condition avec `WHERE` qui porte sur un booléen.

7.1 Les **opérateurs de comparaison** permettent de définir des booléens et les **opérateurs logiques** de calculer sur les booléens.

Opérateurs de comparaison		Opérateurs logiques	
Égalité	=	Conjonction (intersection)	`AND`
Différence	<> ou !=	Disjonction inclusive (union)	`OR`
Inégalité stricte	< ou >	Négation	`NOT`
Inégalité large	<= ou >=	Différence ensembliste	`AND NOT`

7.2 L'opérateur logique `BETWEEN` permet de vérifier qu'un attribut prend une valeur comprise (au sens large) entre deux valeurs particulières.

7.3 L'opérateur logique `IN` permet de vérifier qu'un attribut prend une valeur appartenant à une liste déterminée. Ainsi les deux requêtes suivantes sont équivalentes.

```
SELECT champ_A, champ_B FROM table_A
WHERE champ_A IN (a, b, c)

SELECT champ_A, champ_B FROM table_A
WHERE champ_A = a OR champ_A = b OR champ_A = c
```

7.4 L'opérateur logique `LIKE` permet de filtrer des chaînes de caractères selon un modèle plus ou moins défini à l'aide des caractères spéciaux `%` et `_`.

1. Le caractère _ remplace n'importe quel caractère.

La chaîne `'Gr_goire'` désigne donc les chaînes `'Grégoire'` (avec accent) et `'Gregoire'` (sans accent), mais aussi `'Gragoire'`, `'Grbgoire'`... qu'on s'attend moins à trouver dans une liste de prénoms.

2. Le caractère `%` remplace n'importe quelle chaîne de caractères, y compris la chaîne vide.

La chaîne `'SAINT%'` permet de trouver toutes les communes dont le nom commence par 'Saint' ou 'Sainte' (comme Saint-Vaast-La-Hougue ou Sainte-Adresse), mais aussi Saintines (dans l'Oise).
La chaîne `'%GU%'` permet de trouver toutes les communes dont le nom contient la chaîne de caractères 'gu', soit au début (Guérande), soit au milieu (Martigues), soit à la fin (Montaigu).

8. Tri des enregistrements

8.1 Une table peut être triée par ordre croissant selon l'attribut `champ_A` au moyen de la commande `ORDER BY`.

```
SELECT champ_A, champ_B FROM table_A
ORDER BY champ_A
```

Elle peut aussi être triée par ordre décroissant.

```
SELECT champ_A, champ_B FROM table_A
ORDER BY champ_A DESC
```

8.2 On peut aussi trier les données selon plusieurs attributs avec un analogue de l'ordre lexicographique. La requête suivante retourne une table ordonnée en croissant selon la valeur de l'attribut `champ_A`, les enregistrements dont l'attribut `champ_A` a la même valeur étant ordonnés en décroissant selon la valeur de l'attribut `champ_C`.

```
SELECT champ_A, champ_B, champ_C FROM table_A
ORDER BY champ_A, champ_C DESC
```

8.3 On peut choisir de ne retourner que les `n` premiers résultats d'une requête.

```
SELECT champ_A FROM table_A
LIMIT n
```

On peut aussi omettre les `m` premiers résultats, c'est-à-dire les résultats de rang $0, 1, \ldots, (m-1)$, et afficher les `n` résultats suivants.

```
SELECT champ_A FROM table_A
LIMIT m, n
```

9. Renommage

9.1 On peut renommer un attribut qui apparaît dans le résultat d'une requête avec `AS`.

```
SELECT champ_A AS NouveauNom FROM Table_A
```

9.2 Le renommage est particulièrement utile quand on crée de nouveaux attributs, dont les valeurs sont calculées au moyen des enregistrements sélectionnés.

```
SELECT Vitesse*Durée AS Distance FROM Table_A
```

9.3 On peut aussi, avec `AS`, donner un nom temporaire à une table (quelle que soit sa taille : elle peut être réduite à une seule colonne, une seule ligne, voire une seule valeur), le temps d'effectuer une requête sur les enregistrements de cette table. →[14.4]

Exemples de requêtes

Q 1. *Sélectionner les joueurs de nationalité française (code de nationalité* `FR`*).*

Q 2. *Sélectionner les joueurs qui n'ont pas joué de la saison (temps de jeu nul).*

Q 3. *Sélectionner les joueurs et leur taille, en classant les enregistrements par ordre croissant de taille.*

Q 4. *L'**indice de masse corporelle** (IMC) est le quotient M/L^2 où M est le poids (en kilogrammes) et L est la taille (en mètres).*
Sélectionner les joueurs et calculer leur IMC.

Q 5. *Sélectionner les identités et les tailles respectives des joueurs dont la taille est comprise entre* 180 *et* 190 *centimètres.*

Q 6. *Sélectionner les joueurs nés dans les années 1990 et leurs dates de naissances respectives.*

Q 7. *Sélectionner les identités et les poids respectifs des joueurs qui pèsent moins de* 100 *kilogrammes et qui ont disputé au moins un match avec leur équipe nationale (joueurs dits* internationaux*).*

Q 8. *Lors de la* phase régulière *du Championnat de France (Top 14), chaque équipe affronte toutes les autres équipes deux fois : une fois à domicile, une fois à l'extérieur.*
Lors de la phase éliminatoire*, les matchs ont lieu sur terrain neutre.*
Sélectionner les équipes affrontées lors de la phase régulière du Championnat de France, ainsi que le score de chaque match.

Q 9. *La première phase de la Coupe d'Europe (Champions Cup) est constituée de six journées (J1 à J6).*
Sélectionner les équipes affrontées lors de la première phase de la Coupe d'Europe, ainsi que le résultat `Delta` *de chaque match, calculé comme la différence entre le nombre de points marqués par Clermont et le nombre de points marqués par l'équipe adverse.*

Q 10. *On cherche à déterminer le joueur le plus âgé avec la requête suivante.*

```
SELECT Joueur FROM Effectif
ORDER BY "Né le"
```

Comme chacun sait, le joueur le plus âgé de l'ASM est le très illustre Aurélien Rougerie. Que se passe-t-il ? Comment y remédier ?

III

Fonctions d'agrégation

10. Les **fonctions d'agrégation** sont des fonctions, en général de nature statistique, qu'on applique aux enregistrements d'une table.

10.1 La fonction `COUNT` retourne le nombre d'enregistrements qui vérifient une condition donnée.

10.2 La fonction `SUM` calcule la somme des valeurs prises par les enregistrements qui vérifient une condition donnée.
De même, la fonction `AVG` (pour *average*) calcule la valeur moyenne des enregistrements.

10.3 Les fonctions `MIN` et `MAX` calculent respectivement la plus petite et la plus grande valeur des enregistrements qui vérifient une condition donnée.

11. Fréquemment, on n'applique pas ces fonctions à la totalité des enregistrements, mais aux enregistrements regroupés (avec l'instruction `GROUP BY`) en fonction d'un critère.

```
SELECT COUNT(*) FROM table_A
WHERE Condition
GROUP BY Critère
```

12. Il est possible de grouper les enregistrements en fonction de plusieurs attributs. →[0]

Exemples de requête

Q 11. *Calculer le nombre d'internationaux dans l'effectif de l'ASM.*

Q 12. *Pour chaque nationalité, afficher le nombre d'internationaux dans l'effectif de l'ASM.*

Q 13.a *Calculer le temps de jeu moyen des joueurs de l'ASM.*

Q 13.b *Calculer le temps de jeu moyen en ne tenant compte que des joueurs qui ont effectivement joué.*

Q 13.c *Calculer le temps de jeu moyen des internationaux.*

Q 13.d *Calculer le temps de jeu moyen des joueurs qui ont effectivement joué, d'une part pour les internationaux, d'autre part pour les joueurs qui ne sont pas internationaux.*

Q 14. *Lors de la saison, l'ASM a disputé 37 matchs. Chaque match dure 80 minutes et il y a* au plus *15 joueurs sur le terrain. Il arrive qu'une équipe joue avec moins de 15 joueurs (en cas d'exclusion, temporaire ou définitive, ou de blessure légère).*
Calculer la durée totale pendant laquelle l'ASM a joué avec moins de 15 joueurs.

Q 15. *Calculer la taille moyenne (en centimètres) des joueurs, en fonction de leur nationalité.*

Q 16. *Pour chaque nationalité, calculer le nombre de joueurs, la taille (en centimètres) du plus grand et la taille (en centimètres) du plus petit.*

IV

Sous-requêtes

13. Une **sous-requête** est une requête qui figure à l'intérieur d'une requête.

14. Règles pour la composition des requêtes
Pour que la valeur de $(g \circ f)(x)$ soit définie, il faut d'abord que x appartienne à l'ensemble de définition de f et ensuite que $f(x)$ appartienne à l'ensemble de définition de g.
On peut imbriquer des requêtes exactement comme on compose des fonctions :
14.1 Une première requête, ou *sous-requête*, produit une table, dite **table intermédiaire** ;
14.2 La seconde requête, ou *requête principale*, porte sur des attributs de la table intermédiaire.
14.3 La sous-requête doit être rédigée entre parenthèses.
14.4 Il est bon (et parfois obligatoire) de donner un nom à la table intermédiaire. C'est en particulier nécessaire quand des attributs de deux tables différentes ont le même identifiant.

15. Conditions sur une table intermédiaire
On peut effectuer une sous-requête pour produire une table intermédiaire qui contient des valeurs de référence.
15.1 Si la table intermédiaire ne contient qu'une seule valeur (c'est-à-dire un seul attribut et un seul enregistrement), on peut utiliser cette valeur comme une valeur de référence pour l'attribut considéré.

15.2 Si la table intermédiaire est constituée d'une seule colonne (c'est-à-dire un seul attribut et plusieurs enregistrements), on peut utiliser les valeurs qui figurent dans cette colonne comme valeurs de référence.

1. Avec `IN`, on sélectionne les enregistrements dont la valeur fait partie des valeurs de référence.
2. Avec `NOT IN`, on sélectionne les enregistrements dont la valeur ne fait pas partie des valeurs de référence.

16. Créer une table auxiliaire

Les esprits cartésiens, épris de clarté et de simplicité, éviteront autant que possible d'imbriquer les requêtes.

16.1 Lorsque c'est possible, on peut créer une table avec le résultat d'une première requête en faisant précéder le code de cette requête par le code suivant.

```
CREATE TABLE table_aux AS
```

(Bien entendu, on choisira un nom pertinent pour la table ainsi créée.)
On exécutera ensuite une seconde requête sur cette table auxiliaire, de laquelle on pourra ensuite se débarrasser avec le code suivant.

```
DROP TABLE table_aux
```

16.2 On parvient ainsi au résultat voulu avec deux requêtes simples et séparées, ce qui conduit à un code plus lisible qu'avec un sous-requête rédigée à l'intérieur de la requête principale.

Exemples de requêtes

Q 17. *On considère qu'un IMC* **[Q4.]** *compris entre* 30 *et* 35 *est le signe d'une* obésité modérée*; qu'un IMC supérieur à* 35 *est le signe d'une* obésité morbide*.*
Compter le nombre de joueurs qui seraient en situation d'obésité modérée selon ce critère. (Cette statistique est notoirement inadéquate pour les sportifs.)

Q 18. *Sélectionner les 10 plus grands joueurs de l'effectif en les classant par ordre croissant de taille.*

Q 19. *Pour chaque nationalité représentée par deux joueurs au moins dans l'effectif de l'ASM, calculer le nombre de joueurs, la taille du plus grand joueur et la taille du plus petit joueur. Les résultats seront présentés de la nationalité la moins représentée à la nationalité la plus représentée.*

Q 20. *Sélectionner les joueurs dont le poids est égal au poids moyen de l'effectif. On arrondira le poids moyen de l'effectif avec* `ROUND`*.*

Q 21. *Sélectionner les joueurs internationaux dont la taille est inférieure à la taille moyenne des joueurs internationaux.*

V

Requêtes sur plusieurs tables

V.1 Opérations ensemblistes sur les relations

17. Les opérations ensemblistes (union, intersection et différence) n'ont de sens que si elles portent sur des relations qui ont exactement les mêmes attributs.

18. L'**union** de deux relations consiste à rassembler dans une même table tous les enregistrements qui appartiennent à l'une ou l'autre des relations.

```
SELECT *
FROM table_A UNION table_B
```

Chaque enregistrement n'apparaît qu'une seule fois dans l'union des deux relations.

19. On cherche dans quelle mesure la durée pendant laquelle l'ASM n'a pas joué à 15 (calculée en **[Q14.]**) peut s'expliquer par les exclusions de joueurs.
Au moyen de deux sous-requêtes, on calcule la durée totale des exclusions temporaires (carton jaune) et la durée totale des exclusions définitives (carton rouge). Il faut noter ici qu'une exclusion temporaire dure moins de 10 minutes lorsqu'un joueur est sanctionné après la 70-ième minute du match !

```
SELECT SUM(DuréeExclu) AS "DuréeTotaleExclu"
FROM (
      SELECT rowid,
             MIN(80, 10+"Minute de jeu")-"Minute de jeu" AS DuréeExclu
      FROM Cartons
      WHERE Carton="J"
UNION
      SELECT rowid,
             80-"Minute de jeu" AS DuréeExclu
      FROM Cartons
      WHERE Carton="R"
)
```

Pour que l'union des deux tables ne fasse pas disparaître de données, il faut que les enregistrements soient deux à deux distincts. C'est la raison pour laquelle la clé primaire `rowid` apparaît dans les sous-requêtes.

20. L'**intersection** de deux relations consiste à rassembler dans une table tous les enregistrements qui appartiennent aux deux relations.

```
-- Joueurs ayant été titularisés aux deux postes de deuxième ligne
SELECT * FROM
     (SELECT idJoueur FROM Compositions
      WHERE Poste=4
INTERSECT
      SELECT idJoueur FROM Compositions
      WHERE Poste=5)
```

21. La **différence** de deux relations consiste à rassembler dans une table tous les enregistrements qui appartiennent à la première relation sans appartenir à la seconde relation.

```
-- Joueurs qui ont été retenus comme remplaçants mais jamais comme titulaires
SELECT * FROM
     (SELECT idJoueur FROM Compositions
      WHERE Poste>=16
EXCEPT
      SELECT idJoueur FROM Compositions
      WHERE Poste<=15)
```

V.2 Produit cartésien

22. Le produit cartésien de deux ensembles A et B, noté $A \times B$, est l'ensemble des couples (a, b) lorsque a parcourt A et b parcourt B. En particulier, le cardinal de $A \times B$ est le produit du cardinal de A par le cardinal de B.

23. Le **produit cartésien** de deux relations `table_A` et `table_B` est l'ensemble des enregistrements obtenus en concaténant tous les enregistrements de `table_A` avec chaque enregistrement de `table_C`. Il est obtenu par la commande suivante.

```
table_A, table_B
```

23.1 Il est rare que la concaténation de deux enregistrements ait un sens. Par ailleurs, si les relations dont on fait le produit comptent déjà un nombre important d'enregistrements, leur produit cartésien sera d'une taille colossale !

23.2 Si la relation `table_B` ne contient qu'une seule valeur (le résultat d'une sous-requête), le produit cartésien de `table_A` avec `table_B` consiste à ajouter un attribut à chaque enregistrement de `table_A`, cet attribut ayant *toujours* la même valeur. On peut douter de l'intérêt d'une telle manœuvre, illustrée par la requête suivante. →**[Q22.]**

```
SELECT Joueur
FROM Effectif, (SELECT AVG(Poids) AS Poids_moy,
                       AVG(Taille) AS Taille_moy
                FROM Effectif) AS Moyennes
WHERE Poids<Poids_moy AND Taille>Taille_moy
```

23.3 Cependant, un produit cartésien permet de simplifier le calcul du [**19**].
Une première requête compte le nombre de cartons jaunes qui se sont traduits par une exclusion de 10 minutes.
Une seconde requête calcule la durée totale des exclusions qui ont duré jusqu'au terme de la rencontre, soit à la suite d'un carton rouge, soit à la suite d'un carton jaune donné dans les dix dernières minutes.
La dernière requête porte sur le produit cartésien des deux tables ainsi produites. Ces deux tables ne produisent qu'une seule valeur, le produit cartésien est donc réduit à un seul couple.

```
SELECT exclu_temp+exclu_def AS durée_exclu
FROM
    (SELECT 10*COUNT(*) AS exclu_temp
     FROM Cartons
     WHERE Carton="J" AND "Minute de jeu"<70),
    (SELECT SUM(80-"Minute de jeu") AS exclu_def
     FROM Cartons
     WHERE Carton="R" OR "Minute de jeu">=70)
```

V.3 Jointures internes

24. Mais qui sont les brutes ?
La relation `Cartons` permet de connaître les joueurs qui ont reçu des cartons jaunes pour des fautes graves ou répétées (ou les deux...).

24.1 La requête suivante compte le nombre de cartons jaunes reçus par chacun des joueurs sanctionnés. Le résultat est trié par ordre décroissant du nombre de cartons : le premier enregistrement fait apparaître le joueur le plus souvent sanctionné.

```
SELECT COUNT(*) AS NbCartonsJ, idJoueur
FROM Cartons
WHERE Carton="J"
GROUP BY idJoueur
ORDER BY NbCartonsJ DESC
```

24.2 La table produite par la requête précédente est malheureusement anonyme : on connaît les identifiants des joueurs sanctionnés, mais pas leurs identités. En effet, leurs identités apparaissent dans la relation `Effectif` mais pas dans la relation `Cartons` (pour ne pas dupliquer des informations). L'opération de **jointure** permet de relier les informations contenues dans deux tables au moyen d'un critère particulier : la **condition de jointure**.

```
SELECT COUNT(*) AS NbCartonsJ, Joueur
FROM Cartons     -- table de gauche
JOIN Effectif    -- table de droite
ON Cartons.idJoueur=Effectif.idJoueur  -- condition de jointure
WHERE Carton="J"
GROUP BY Cartons.idJoueur
ORDER BY NbCartonsJ DESC
```

24.3 Formellement, une jointure sur les relations A et B avec la condition de jointure `cond` consiste à effectuer la sélection décrite par `cond` sur le produit cartésien $A \times B$.
En pratique, on peut imaginer une jointure comme la juxtaposition de deux tables, les lignes de la seconde table prolongeant les lignes de la première en fonction de la condition de jointure. On a tout intérêt à se représenter une jointure de la manière suivante (on n'indique que les principaux attributs).

Cartons	
Carton	idJoueur
"J"	28
"J"	46
"J"	19
⋮	⋮

Effectif	
idJoueur	Joueur
1	Abendanon
2	Astier
3	Bardy
⋮	⋮

Cartons ⋈ Effectif			
Carton	idJoueur	idJoueur	Joueur
"J"	28	28	Lee
"J"	46	46	Stanley
"J"	19	19	Grosso
⋮	⋮	⋮	⋮

Une fois que cette table (virtuelle) est construite, on peut y effectuer n'importe quelle requête.
24.4 Dans cet exemple, la condition de jointure est très simple : les relations `Cartons` et `Effectif` ont un attribut en commun (l'attribut `idJoueur`) et la condition de jointure demande que cet attribut prenne la même valeur dans les deux tables. Dans ce cas, on peut simplifier l'écriture de la requête.

```
SELECT COUNT(*) AS NbCartonsJ, Joueur
FROM Cartons            -- table de gauche
NATURAL JOIN Effectif   -- table de droite et condition naturelle de jointure
WHERE Carton="J"
GROUP BY Cartons.idJoueur
ORDER BY NbCartonsJ DESC
```

Q 22. *Sélectionner les joueurs dont le poids est inférieur au poids moyen et dont la taille est supérieure à la taille moyenne.*

Q 23. *Pour chaque nationalité, calculer la proportion de joueurs internationaux dans l'effectif de l'ASM.*

Q 24. *Dresser la liste des joueurs qui ont été titularisés au moins 8 fois dans un poste donné au cours de la saison et des postes auxquels ils ont été titularisés.*

Q 25. *Pour chaque joueur, calculer le total des points marqués et calculer l'efficacité (c'est-à-dire le quotient du nombre de points marqués par le temps de jeu). On prendra pour unité de temps la durée d'un match, c'est-à-dire 80 minutes.*

Réponses aux questions

II Requêtes simples

R 1.

```
SELECT Joueur FROM Effectif
WHERE Nationalité='FR'
```

R 2.

```
SELECT Joueur FROM Effectif
WHERE "Temps de jeu"=0
```

R 3.

```
SELECT Joueur, Taille
FROM Effectif
ORDER BY Taille
```

R 4. La formule de l'IMC est donnée en mètres, les tailles des joueurs sont connues en centimètres : il faut convertir !
Pour éviter de diviser par 0, on restreint la sélection aux enregistrements qui correspondent à des joueurs humains (on rappelle la présence dans la table d'un joueur fictif qui sert d'attribut aux essais de pénalité).

```
SELECT Joueur, Poids/(0.0001*Taille*Taille) AS IMC
FROM Effectif
WHERE Taille NOT NULL
```

On conseille vivement, dès que la requête devient un peu complexe, de mettre en évidence la structure logique de la requête (en allant à la ligne par exemple).

R 5.

```
SELECT Joueur, Taille
FROM Effectif
WHERE Taille BETWEEN 180 AND 190
```

R 6.

```
SELECT Joueur,
       "Né le" AS DDN
FROM Effectif
WHERE "Né le" LIKE "%199%"
```

R 7.

```
SELECT Joueur, Poids
FROM Effectif
WHERE Poids<100 AND International
```

R 8.

```
SELECT Adversaire, "Points Montferrand", "Points adversaire"
FROM Matchs
WHERE (Terrain="1" OR Terrain="2") AND (Compétition="Top 14")
```

R 9.

```
SELECT Adversaire,
       "Points Montferrand"-"Points adversaire" AS Delta
FROM Matchs
WHERE Compétition="Champions Cup" AND Niveau LIKE "J%"
```

R 10. Les dates de naissance n'ont pas été triées en fonction de l'année, mais en fonction du quantième ! En effet, elles ont été inscrites sous forme de chaînes de caractères et non de dates : le bon format est "AAAA/MM/JJ" (comme dans la table `Matchs`) et non pas "JJ/MM/AAAA". Il faut donc remettre les dates au bon format.
Voici la marche à suivre en pareil cas.

1. On exporte la table `Effectif` dans un fichier `EffectifA.csv`, puis on supprime cette table de la base de données.
2. On applique le script Python suivant pour créer un fichier `EffectifB.csv`.

```
effectif = open('EffectifA.csv', 'r')  # fichier source, ouvert en lecture
effectifB = open('EffectifB.csv', 'w') # fichier but, ouvert en écriture
effectifB.write(effectif.readline())   # On recopie l'en-tête tel quel.

data = effectif.readline()
while (data!=''):                      # Pour chaque enregistrement:
    # On énumère les attributs dans une liste.
    liste_data = data.split(";")
    # On extrait la date de naissance de cette liste (attribut de rang 3)
    # et on supprime les guillemets qui l'encadrent.
    DDN = liste_data[3][1:-1]
    if (len(DDN)>0):
        # On récupère les trois composantes de la date de naissance
        jj, mm, aaaa = DDN.split("/")
        # et on réécrit la date de naissance au bon format.
        liste_data[3] = "/".join([aaaa, mm, jj])
        # On écrit l'enregistrement modifié dans le fichier but.
        effectifB.write(";".join(liste_data))
    else:
        # L'enregistrement du joueur fictif est recopié tel quel.
        effectifB.write(data)
    # On passe à l'enregistrement suivant.
    data = effectif.readline()
effectif.close()                       # fermeture du fichier source
effectifB.close()                      # fermeture du fichier but
```

Si le fichier but n'est pas fermé, il risque de ne pas être écrit sur le disque dur !

3. On importe le nouveau fichier dans la base de données pour recréer la table `Effectif`.

Remarque. Lorsque les dates sont écrites au format `AAAA/MM/JJ`, le tri des dates par ordre chronologique coïncide avec le tri par ordre lexicographique. Ainsi, on n'a pas besoin d'un algorithme spécifique pour le tri des dates !

III Fonctions d'agrégation

R 11. On peut *compter* le nombre d'enregistrements pour lesquels le booléen est égal à 1.

```
SELECT COUNT(*) AS NbInternx
FROM Effectif
WHERE International
```

On peut aussi calculer la *somme* des booléens.

```
SELECT SUM(International) AS NbInternx
FROM Effectif
```

R 12. On regroupe les joueurs par nationalité en éliminant le joueur fictif (pour éviter d'introduire un enregistrement absurde dans la table qui résulte de la requête).

```
SELECT Nationalité,
       COUNT(International) AS NbInternx
FROM Effectif
WHERE Nationalité NOT NULL
GROUP BY Nationalité
```

R 13.a

```
SELECT AVG("Temps de jeu") AS "Temps de jeu moy"
FROM Effectif
```

R 13.b

```
SELECT AVG("Temps de jeu") AS "Temps de jeu moy"
FROM Effectif
WHERE "Temps de jeu">0
```

R 13.c

```
SELECT AVG("Temps de jeu") AS "Temps de jeu moy"
FROM Effectif
WHERE International
```

R 13.d

```
SELECT AVG("Temps de jeu") AS "Temps de jeu moy"
FROM Effectif
WHERE "Temps de jeu">0
GROUP BY International
```

R 14.

```
SELECT 37*80*15 - SUM("Temps de jeu") AS "Durée jouée en infériorité"
FROM Effectif
```

R 15. On prend encore soin d'éliminer le joueur fictif du résultat.

```
SELECT Nationalité,
       AVG(Taille) AS TailleMoy
FROM Effectif
WHERE Nationalité NOT NULL
GROUP BY Nationalité
ORDER BY TailleMoy
```

R 16. On élimine cette fois le joueur fictif en ne prenant en compte que les joueurs dont la taille est non nulle.

```
SELECT Nationalité,
       COUNT(idJoueur) AS Nbre,
       MIN(Taille) AS TailleMin,
       MAX(Taille) AS TailleMax
FROM Effectif
WHERE Taille>0
GROUP BY Nationalité
ORDER BY Nbre
```

IV Sous-requêtes

R 17. On effectue les calculs sur les enregistrements de la table produite par la requête de **[R4.]**. Il est utile, pour une bonne lisibilité du code, de mettre en évidence la sous-requête.

```
SELECT COUNT(Joueur) AS "Obésité modérée" FROM
    (SELECT Joueur, Poids/(0.0001*Taille*Taille) AS IMC
     FROM Effectif
     WHERE Taille NOT NULL)
WHERE IMC BETWEEN 30 AND 35
```

On voit sur cet exemple l'utilité d'un alias lors du calcul d'un nouvel attribut.

R 18. Avec une première requête, on constitue la table des 10 plus grands joueurs classés par ordre *décroissant*. Une seconde requête permet de réorganiser cette table par ordre *croissant*.

```
SELECT * FROM
    (SELECT idJoueur, Taille FROM Effectif
     ORDER BY Taille DESC
     LIMIT 10) AS Grands_joueurs
ORDER BY Taille
```

R 19. Une première requête calcule les valeurs cherchées en regroupant les joueurs en fonction de leur nationalité. Une seconde requête élimine le joueur fictif et les nationalités qui ne sont représentées que par un seul joueur et trie les données dans l'ordre demandé.

```
SELECT * FROM (
    SELECT Nationalité,
           COUNT(idJoueur) AS Nbre,
           MIN(Taille) AS TailleMin,
           MAX(Taille) AS TailleMax
    FROM Effectif
    GROUP BY Nationalité)
WHERE Nbre>1
ORDER BY Nbre
```

R 20. Une première requête calcule le poids moyen de l'effectif. La seconde requête sélectionne les joueurs dont le poids coïncide avec le poids moyen calculé précédemment.

```
SELECT Joueur FROM Effectif
WHERE Poids = (SELECT ROUND(AVG(Poids))
               FROM Effectif)
```

R 21. Une première requête calcule la taille moyenne des joueurs internationaux. La seconde requête sélectionne les joueurs internationaux dont la taille est inférieure à la moyenne calculée précédemment.

```
SELECT Joueur FROM Effectif
WHERE International
AND Taille < (SELECT AVG(Taille)
              FROM Effectif
              WHERE International)
```

V Requêtes sur plusieurs tables

R 22. Une sous-requête renvoie une table auxiliaire qui contient le poids moyen et la taille moyenne. En reliant cette table à la table `Effectif` au moyen d'une jointure, on trouve la liste des joueurs dont le poids est inférieur au poids moyen et dont la taille est supérieure à la taille moyenne.

```
SELECT Joueur
FROM Effectif
JOIN
    (SELECT AVG(Poids) AS Poids_moy,
            AVG(Taille) AS Taille_moy FROM Effectif)
    AS Moyennes
ON Poids<Poids_moy AND Taille>Taille_moy
```

Cette requête est sensiblement plus efficace que celle du [**23.2**], puisqu'elle évite de créer une table auxiliaire de grande taille.

R 23. On constitue deux tables intermédiaires :
– La table *A* contient le nombre de joueurs internationaux pour chaque nationalité ;
– La table *B* contient le nombre total de joueurs pour chaque nationalité.
On effectuant une jointure de ces deux tables en fonction de la nationalité des joueurs, on peut récupérer les nationalités, les nombres d'internationaux et les nombres totaux et en déduire les proportions d'internationaux (qu'il est utile de renommer).

```
SELECT A.Nat AS Nationalité,
       (100*NbInternx)/NbTotal AS PropInternx
FROM (
       SELECT Nationalité AS Nat,
              COUNT(*) AS NbInternx
       FROM Effectif
       WHERE International
       GROUP BY Nationalité
) AS A              -- table de gauche
JOIN (
      SELECT Nationalité AS Nat,
             COUNT(*) AS NbTotal
      FROM Effectif
      GROUP BY Nationalité
) AS B              -- table de droite
ON A.Nat = B.Nat    -- condition de jointure
```

Puisque la condition de jointure porte sur la nationalité qui est un attribut commun aux deux tables intermédiaires, on peut proposer une variante allégée avec `NATURAL JOIN`.

```
SELECT Nationalité,
       (100*NbInternx)/NbTotal AS PropInternx
FROM (
       SELECT Nationalité,
              COUNT(*) AS NbInternx
       FROM Effectif
       WHERE International
       GROUP BY Nationalité
) AS A
NATURAL JOIN (
       SELECT Nationalité,
              COUNT(*) AS NbTotal
       FROM Effectif
       GROUP BY Nationalité
) AS B
```

R 24. La requête suivante calcule le nombre de titularisations (numéro de poste inférieur à 15) pour chaque joueur et chaque poste.

```
SELECT idJoueur,
       Poste,
       COUNT(*) AS Titularisations
FROM Compositions
WHERE Poste<=15
GROUP BY Poste, idJoueur
```

Il reste à réaliser une jointure naturelle (sur l'attribut `idJoueur`) avec la relation `Effectif` afin de faire apparaître en clair l'identité de chaque joueur et éliminer les enregistrements qui font apparaître un nombre de titularisations trop faible.

```
SELECT Joueur,
       Poste,
       Titularisations
FROM (  -- table résultant de la requête précédente
      SELECT idJoueur,
             Poste,
             COUNT(*) AS Titularisations
      FROM Compositions
      WHERE Poste<=15
      GROUP BY Poste, idJoueur
      )
NATURAL JOIN Effectif
WHERE Titularisations>=8
ORDER BY Poste
```

R 25. On commence par compter, pour chaque joueur et pour chaque type de points (essai, transformation, pénalité, drop), le nombre total de réalisations et le nombre de points attribués à chaque réalisation. Il faut pour cela réaliser une jointure entre la relation `Points` (qui liste les types de points marqués par les joueurs) et la relation `ValeursPoints` (qui donne le nombre de points attribués aux différents types de points).
La condition de jointure porte sur le type de points marqués. Comme les identifiants de cet attribut sont différents dans les deux relations, il n'est pas possible de passer par un `NATURAL JOIN`.

```
SELECT idJoueur,
       Points,
       COUNT(*) AS nb,
       Valeur
FROM Points
JOIN ValeursPoints
ON Points.Points=ValeursPoints.idPoints
GROUP BY idJoueur, Points
```

On déduit de la requête précédente le nombre total de points marqués pour chaque joueur.

```
SELECT idJoueur,
       SUM(nb*Valeur) AS TotalPoints
FROM (
      -- première requête
     )
GROUP BY idJoueur
```

On peut maintenant regrouper dans un même enregistrement l'identité du joueur, le nombre total de points marqués ainsi que le temps total de jeu au moyen d'une jointure naturelle (portant sur l'attribut `idJoueur`) avec la relation `Effectif`.

```
SELECT Joueur,                -- relation Effectif
       TotalPoints,           -- résultat de la deuxième requête
       "Temps de jeu" AS TJ   -- relation Effectif
FROM (
      -- deuxième requête
     )
NATURAL JOIN Effectif
```

On en déduit enfin les valeurs cherchées, en prenant en compte que le temps de jeu total est donné en minutes alors que l'unité de temps pour le calcul de l'efficacité est la durée normale d'une rencontre (80 minutes).

```
SELECT Joueur,
       TotalPoints,
       80*TotalPoints/TJ AS Efficacité
FROM (
      -- troisième requête
      )
WHERE TJ>0
ORDER BY Efficacité DESC
```