5
L'Académie Française

1. Création de l'Académie

De 1624 à sa mort en 1642, Armand Jean du Plessis, cardinal-duc de Richelieu, fut le principal Ministre d'État de Louis XIII. Il a créé l'Académie française en 1635 pour *travailler, avec tout le soin et toute la diligence possibles, à donner des règles certaines à notre langue et à la rendre pure, éloquente et capable de traiter les arts et les sciences* (article 24 des statuts).

Si François I^{er} avait fait du français la langue administrative et judiciaire commune (Édit de Villers-Cotterêts, 1539), la langue française était encore flottante et variable un siècle plus tard. Fixer la langue, rendre son usage plus précis et plus universellement compréhensible furent pour le pouvoir royal des *instruments de sa politique d'unification du royaume à l'intérieur et de son rayonnement diplomatique à l'étranger* [1].

2. Vicissitudes de l'histoire

Le 8 août 1793, la Convention supprime par décret toutes les académies royales, dont l'Académie française. Un Institut national des sciences et des arts les remplace en 1795.

Un arrêté du 23 février 1803 crée une section de l'Institut consacrée à la Langue et à la Littérature française, constituée de 40 membres.

Le 21 mars 1816, la Restauration rétablit les appellations traditionnelles et exclut quelques membres.

Au vingtième siècle, les guerres mondiales ont aussi troublé les activités de l'Académie. →**[Q18.]**

3. Vocabulaire des bases de données

3.1 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.

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

3.3 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.

1. `http://academie-francaise.fr/linstitution/les-missions`

I

La base de données AcadémieFrançaise.sqlite

4. La base de données `AcadémieFrançaise.sqlite` a été établie à partir de données collectées sur Wikipedia. Elle est constituée de deux tables.

4.1 La table `Académiciens` regroupe les données d'état-civil des membres de l'Académie.

Attributs de la table `Académiciens`		
`id_acad`	`Integer`	Clé primaire
`Sexe`	`Integer`	1 pour les hommes 2 pour les femmes
`Nom`	`Text`	Nom (nom d'usage ou pseudonyme éventuel)
`DDN`	`Varchar`	Date de naissance `AAAA-MM-JJ`
`DDD`	`Varchar`	Date de décès `AAAA-MM-JJ` (`NULL` pour les vivants)

4.2 La table `Fauteuils` contient les informations propres à l'Académie.

Attributs de la table `Fauteuils`		
`Fauteuil`	`Integer`	Numéro de fauteuil
`id_acad`	`Integer`	Identifiant de l'académicien
`Election`	`Varchar`	Date d'élection à l'Académie `AAAA-MM-JJ`
`Exclusion`	`Varchar`	Date d'exclusion `AAAA-MM-JJ`
`Motif`	`Text`	Circonstances de l'exclusion

5. Cette base de données sert à s'exercer.

5.1 Il doit paraître clair que toute l'information nécessaire au calcul de chaque exemple est contenue dans la base, il ne s'agit donc que de formuler la requête qui va permettre d'effectuer ce calcul autrement qu'en comptant sur ses doigts.

5.2 Comme pour la plupart des bases de données, certaines données sont manquantes et impossibles à compléter : l'état-civil du dix-septième siècle n'était pas ce qu'il est ! Ces lacunes doivent inciter à considérer avec prudence certains résultats calculés.

5.3 Il est probable que les données de cette base soient exactes. Néanmoins, la prudence et l'honnêteté exigent qu'on se soucie de la qualité des données traitées : si les données sont inexactes, le résultat de leur traitement le sera aussi.

5.4 Indépendamment de la fiabilité des données à traiter, il convient aussi, « *pour l'honneur de l'esprit humain* », de se soucier du sens des résultats calculés [2]. Les exemples traités ici échappent à cette règle, puisqu'il ne s'agit que d'exercices d'ordre scolaire.

Q 1. *Pourquoi écrit-on les dates au format* `AAAA-MM-JJ` *plutôt qu'au format* `JJ-MM-AAAA`*, qui serait plus naturel ?*

2. Les amateurs d'absurde pourront consulter `http://tylervigen.com/spurious-correlations`

II

Requêtes simples

6. Sélection

La **sélection**, ou **restriction**, consiste à extraire d'une relation `table_A` les enregistrements qui vérifient un certain critère. On note : $\sigma_{\text{critère}}(\texttt{table\_A})$.
Pour éviter d'avoir à écrire la liste des tous les attributs, on utilise le caractère spécial $*$.

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

7. Projection

Une **projection** consiste à extraire une partie des attributs d'une relation. La projection qui extrait les attributs $A_1, \ldots, A_n$ de la relation `table_A` est notée $\pi_{A_1,\ldots,A_n}(\texttt{table\_A})$.

7.1 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
```

7.2 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
```

7.3 Suppression des doublons

Il ne peut y avoir de doublons dans une relation, mais des doublons peuvent apparaître après une projection. Si on le souhaite, on peut demander que les doublons soient supprimés.

```
SELECT DISTINCT champ_A, champ_B
FROM table_A
```

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

8.1 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

8.2 L'opérateur logique `IS NULL` permet de vérifier qu'un attribut n'a pas de valeur.

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

8.4 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
```

8.5 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).

3. Si `ch` est une chaîne de caractères, la fonction `SUBSTR(ch, départ, nb)` retourne la sous-chaîne constituée de `nb` caractères à partir du caractère d'indice `départ`, l'indice du premier caractère de la chaîne étant égal à 1.

Par exemple, `SUBSTR("Séguier", 1, 3)` retourne `"Ség"`.

9. Tri des enregistrements

9.1 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
```

9.2 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
```

9.3 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
```

10. Renommage

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

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

10.2 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
```

11. 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. →[**15.4**]

Exemples de requêtes

Q 2. *Comment s'appelle l'opération effectuée par la requête suivante ?*

```
SELECT Sexe, Nom, DDN FROM Académiciens
```

Q 3. *Quelle est l'opération qui permet d'obtenir* `id_acad`, `Sexe`, `Nom`, `DDN` *et* `DDD` *pour les académiciens nés au cours du vingtième siècle ? Traduire cette opération en requête SQL.*

Q 4. *Sélectionner les académiciens nés dans les années 1870.*

Q 5. *Que calcule la requête suivante ?*

```
SELECT id_acad, SUBSTR(DDN, 6, 2) AS Mois_Naiss FROM Académiciens
WHERE Mois_Naiss!="xx"
ORDER BY Mois_Naiss
```

III

Fonctions d'agrégation

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

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

12.2 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.

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

13. 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
```

Il est possible de grouper les enregistrements en fonction de plusieurs attributs. →**[Q20.]**

Exemples de requêtes

Q 6. *Compter le nombre d'académiciens vivants de chaque sexe.*

Q 7. *Compter, pour chaque fauteuil, le nombre d'académiciens qui l'ont occupé. Trier les résultats pour faire apparaître les fauteuils qui ont connu le moins d'occupants.*

Q 8.a *Quels sont les calculs effectués par la requête suivante ?*

```
SELECT Nom, DDN, DDD, DDD-DDN AS Âge FROM Académiciens
WHERE DDN LIKE "187%"
```

Q 8.b *En déduire le moyen de calculer l'âge moyen de décès pour les académiciens nés dans la décennie 1900-1909.*

Q 9. *Que calcule la requête suivante ?*

```
SELECT DDN, COUNT(*) AS Nbre FROM Académiciens
GROUP BY DDN
ORDER BY Nbre DESC
```

L'attribut `DDN` *peut-il servir de clé primaire dans la table* `Académiciens` *?*

IV

Sous-requêtes

14. Une **sous-requête** est une requête qui figure à l'intérieur d'une requête.
Le résultat produit par la sous-requête est une table intermédiaire qui réunit toute l'information nécessaire à l'exécution de la requête principale.

15. 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 :
15.1 Une première requête, ou *sous-requête*, produit une table, dite **table intermédiaire** ;
15.2 La seconde requête, ou *requête principale*, porte sur des attributs de la table intermédiaire.
15.3 La sous-requête doit être rédigée entre parenthèses.
15.4 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.

16. Conditions sur une table intermédiaire
On peut effectuer une sous-requête pour produire une table intermédiaire qui contient des valeurs de référence.

16.1 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.
16.2 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.

17. Créer une table auxiliaire
Les esprits cartésiens, épris de clarté et de simplicité, éviteront autant que possible d'imbriquer les requêtes.
17.1 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.
17.2 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
```

17.3 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 10. *On cherche les doyens d'âge de chaque sexe parmi les académiciens vivants.*
Q 10.a *On a besoin de connaître le nom, le sexe et la date de naissance (projection) de tous les académiciens vivants (sélection). Écrire une requête pour obtenir ces informations.*
Q 10.b *L'académicien le plus âgé est celui dont la date de naissance est la plus petite. Comme on cherche les doyens pour chaque sexe, les enregistrements doivent être regroupés en fonction de l'attribut* `sexe`*. Écrire une requête qui effectue ces opérations.*
Q 11. *Compter les académiciens selon leur mois de naissance.*
Q 12. *On s'intéresse seulement aux académiciens nés entre 1900 et 1999 : la requête suivante montre qu'il y en a 104.*

```
SELECT COUNT(*) FROM Académiciens WHERE DDN LIKE "19%"
```

Certains auraient-ils pu fêter leur anniversaire ensemble ?
Q 13. *Compter le nombre d'élections par académicien. Certains ont-ils été élus plus d'une fois ? Quels sont leurs noms ?*

V

Requêtes sur plusieurs tables

Q 14. *En* **[Q10.]**, *on a identifié les doyens d'âge des deux sexes parmi les académiciens vivants au moyen de la table* `Académiciens`.
On s'intéresse maintenant aux doyens d'élection, c'est-à-dire aux académiciens qui siègent depuis le plus grand nombre d'années. Pourquoi ne peut-on accéder au résultat avec une requête sur une seule table ?

V.1 Opérations ensemblistes sur les relations

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

19. 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. Pour que l'union des deux tables ne fasse pas disparaître de données, il faut donc que les enregistrements soient deux à deux distincts : on peut utiliser la clé primaire (`rowid` par défaut) pour cela dans les sous-requêtes qui établissent les deux tables auxiliaires.

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

```
SELECT * FROM
(table_A INTERSECT table_B)
```

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.

```
SELECT * FROM
(table_A EXCEPT table_B)
```

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
```

V.3 Jointures

24. La table `Fauteuil` permet de connaître les occupants successifs du fauteuil 14, ainsi que la date de leur élection.

```
SELECT id_acad, Election FROM Fauteuils AS F
WHERE Fauteuil=14
```

24.1 Malheureusement, le résultat est peu parlant ! On va donc chercher dans une autre table les noms des académiciens dont on vient de recueillir les identifiants.

```
SELECT A.Nom, F.Election FROM Fauteuils AS F
JOIN Académiciens AS A
ON A.id_acad=F.id_acad
WHERE F.Fauteuil=14
```

24.2 On voit ici l'intérêt de renommer une table : chaque attribut devant être préfixé par le nom de la table, l'usage d'un alias permet d'alléger sensiblement la requête.

25. 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 $A \times B$:

$$A \bowtie_{\text{cond}} B = \sigma_{\text{cond}}(A \times B).$$

25.1 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).

Fauteuils

Fauteuil	id_acad
1	35
14	50
31	70
⋮	⋮

Académiciens

id_acad	Nom
35	Séguier
50	Corneille
70	Furetière
⋮	⋮

Fauteuils ⋈ Académiciens

Fauteuil	id_acad	id_acad	Nom
1	35	35	Séguier
14	50	50	Corneille
31	70	70	Furetière
⋮	⋮	⋮	⋮

Une fois que cette table (virtuelle) est construite, on peut y effectuer n'importe quelle requête.

25.2 Dans nos exemples, la condition de jointure est très simple : les relations `Fauteuils` et `Académiciens` ont un attribut en commun (l'attribut `id_acad`) et la condition de jointure demande que cet attribut prenne la même valeur dans les deux tables.

Exemples de requêtes

Q 15. *Quels sont les noms et numéros de fauteuil des académiciens vivants ?*

Q 16. Fauteuils vacants
On s'intéresse ici aux fauteuils vacants.

Q 16.a *Écrire une requête pour connaître le dernier académicien élu pour chaque fauteuil.*

Q 16.b *En déduire quels sont les fauteuils actuellement vacants.*

Q 17. Les académiciens exclus
Certains académiciens ont été exclus, la plupart du temps lors de temps politiquement troublés, mais pas toujours. Quels sont leurs noms et les dates de leurs exclusions ?

Q 18. Les guerres mondiales

Q 18.a *Quelle était la composition de l'Académie au 1er janvier 1918 ? En déduire quels étaient les fauteuils vacants.*

Q 18.b *Quelle était la composition de l'Académie au 12 octobre 1944 ?*

Q 19. Doyens d'élection

Q 19.a *Quels sont à ce jour les noms des doyens d'élection ? On s'inspirera de la requête du* **[R10.]** *pour calculer une table auxiliaire et de* **[R14.]**.

Q 19.b *Au 1er janvier 1675, l'Académie comptait deux doyens d'élection, élus le même jour. Qui étaient-ils ?*

Q 20. *Pour chaque fauteuil, Compter le nombre d'académiciens ayant siégé dans chaque fauteuil, en distinguant les hommes et les femmes.*

Réponses aux questions

I La base de données AcadémieFrançaise.sqlite

R 1. Le format `AAAA-MM-JJ` est le seul pour lequel l'ordre lexicographique sur les chaînes de caractères coïncide avec l'ordre chronologique sur les dates.

II Requêtes simples

R 2. Cette opération est la *projection* de la relation `Académiciens` selon les trois attributs `Sexe`, `Nom` et `DDN`, soit $\pi_{\text{sexe,Nom,DDN}}(\texttt{Académiciens})$.

R 3. Il s'agit de récupérer tous les enregistrements pour lesquels l'attribut `DDN` prend une valeur strictement comprise entre `"1900-12-31"` et `"2001-01-01"` (naissance entre le 1er janvier 1901 et le 31 décembre 2000).
Cette opération est donc la *sélection* $\sigma_{\text{"1900-12-31"<DDN<"2001-01-01"}}(\texttt{Académiciens})$.

```
SELECT * FROM Académiciens
WHERE DDN>"1900-12-31" AND DDN<"2001-01-01"
```

R 4.

```
SELECT * FROM Académiciens
WHERE DDN LIKE "187%"
```

R 5. La commande `SUBSTR(DDD, 6, 2)` extrait le mois de naissance de la chaîne `DDN`. On récupère alors, pour chaque académicien, son identifiant et le mois de sa naissance lorsqu'il est connu (le mois de naissance est inconnu de la base lorsque l'attribut `DDN` est de la forme `"AAAA-xx-JJ"`.
Les enregistrements de cette nouvelle table sont triés par ordre croissant de mois.

III Fonctions d'agrégation

R 6. Un académicien est vivant lorsque l'attribut `DDD` n'a pas de valeur. On compte alors le nombre d'enregistrements pour lesquels `DDD` est `NULL` en regroupant ces enregistrements en fonction des valeurs prises par l'attribut `Sexe`.

```
SELECT Sexe, COUNT(*) AS Nombre FROM Académiciens
WHERE DDD IS NULL
GROUP BY Sexe
```

R 7.

```
SELECT Fauteuil, COUNT(*) AS NbreAcad FROM Fauteuils
GROUP BY Fauteuil
ORDER BY NbreAcad
```

R 8.a Les valeurs des attributs `DDD` et `DDN` sont des *chaînes de caractères*. On a demandé de calculer leur différence comme s'il s'agissait de *nombres* et l'interpréteur a converti la sous-chaîne des années en nombre : on a ainsi calculé l'âge de décès des académiciens nés dans les années 1870. (Il s'agit d'un âge approximatif, puisqu'on ne tient pas compte de la date anniversaire exacte.)

R 8.b

```
SELECT AVG(DDD-DDN) AS ÂgeMoyen FROM Académiciens
WHERE DDN LIKE "190%"
```

R 9. La requête calcule le nombre d'enregistrements de la table `Académiciens` pour chaque valeur de l'attribut `DDN` — autrement dit, le nombre d'académiciens nés le même jour. La table ainsi produite est triée par ordre décroissant de l'attribut `Nbre`.
On constate que, pour 732 académiciens, il n'y a que 709 valeurs différentes pour l'attribut `DDN`. Or, pour qu'un attribut puisse servir de clé primaire, il faut que chaque valeur de cet attribut caractérise un enregistrement. Ce n'est donc pas le cas ici !
En examinant de plus près la table produite par cette requête, on trouve deux explications :
– D'une part, les dates de naissance sont assez mal connues jusqu'au dix-septième siècle : trois académiciens ont la valeur `1675-xx-xx` pour l'attribut `DDN`.
– D'autre part, on voit sur le résultat de la requête que deux académiciens sont nés le 29 novembre 1785 ; que deux autres sont nés le 5 juillet 1899 ; et que deux autres encore sont nés le 2 août 1941.

IV Sous-requêtes

R 10.a

```
SELECT Sexe, Nom, DDN -- projection
FROM Académiciens
WHERE DDD IS NULL -- sélection des académiciens vivants
```

R 10.b

```
SELECT Sexe, Nom, MIN(DDN) AS DDN
FROM -- table intermédiaire produite par la sous-requête
   (SELECT Sexe, Nom, DDN FROM Académiciens WHERE DDD IS NULL)
GROUP BY Sexe
```

R 11. La table `Académiciens` permet de créer une table qui contient l'identifiant de chaque académicien ainsi que le mois de sa naissance.

```
SELECT id_acad, SUBSTR(DDN, 6, 2) AS Mois_Naiss
FROM Académiciens
WHERE Mois_Naiss!="xx"
```

Il n'y a plus qu'à compter le nombre d'occurences de chaque mois dans cette table auxiliaire.

```
SELECT Mois_Naiss, COUNT(*) AS Nbre FROM
 (SELECT id_acad, SUBSTR(DDN, 6, 2) AS Mois_Naiss
  FROM Académiciens
  WHERE Mois_Naiss!="xx")
GROUP BY Mois_Naiss
ORDER BY Mois_Naiss
```

R 12. On complète la première requête pour constituer une table qui contient l'identifiant de chaque académicien né entre 1900 et 1999 ainsi que la date de son anniversaire.

```
SELECT id_acad, SUBSTR(DDN, 6, 5) AS Anniversaire FROM Académiciens
WHERE DDN LIKE "19%"
```

Il n'y a plus qu'à compter les enregistrements de cette table auxiliaire en les regroupant selon l'attribut `Anniversaire`. On trie les résultats pour faire apparaître en premier les anniversaires les plus fréquents : le 17 juin et le 18 août.

```
SELECT COUNT(*) AS Nbre, Anniversaire FROM
 (SELECT id_acad, SUBSTR(DDN, 6, 5) AS Anniversaire
  FROM Académiciens
  WHERE DDN LIKE "19%")
GROUP BY Anniversaire
ORDER BY Nbre DESC
```

R 13. Il est facile de compter le nombre d'élections par académicien.

```
SELECT id_acad, COUNT(*) AS NbElections FROM Fauteuils
GROUP BY id_acad
```

On extrait de cette table auxiliaire l'identifiant des académiciens élus plus d'une fois.

```
SELECT id_acad FROM
 (SELECT id_acad, COUNT(*) AS NbElections FROM Fauteuils
  GROUP BY id_acad)
WHERE NbElections>1
```

Il n'y a plus qu'à sélectionner les noms de ces académiciens dans la table `Académiciens` (ce qui revient à faire une jointure sans le dire).

```
SELECT Nom FROM Académiciens
WHERE id_acad IN        -- seconde table auxiliaire
 (SELECT id_acad FROM   -- première table auxiliaire
   (SELECT id_acad, COUNT(*) AS NbElections FROM Fauteuils
    GROUP BY id_acad)
  WHERE NbElections>1)
```

V Requêtes sur plusieurs tables

R 14. Nous avons besoin de connaître les académiciens vivants et cette information se trouve dans la table `Académiciens` (attribut `DDD`). Nous avons aussi besoin de connaître la date d'élection de chacun d'eux, cette information se trouve dans la table `Fauteuils`. Il faut donc croiser les données de ces deux tables pour otbenir le résultat.

R 15. Le nom et l'absence de date de décès se trouvent dans la table `Académiciens`, le numéro de fauteuil se trouve dans la table `Fauteuils` : il faut bien réaliser une jointure pour obtenir le résultat.

```
SELECT F.Fauteuil, A.Nom FROM Académiciens AS A
JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE A.DDD IS NULL
```

Fauteuils vacants

R 16.a On cherche la plus grande valeur de l'attribut `Election` en regroupant les enregistrements selon l'attribut `Fauteuil`.

```
SELECT F.Fauteuil AS Fauteuil, MAX(F.Election), A.Nom
FROM Fauteuils AS F
JOIN Académiciens AS A ON A.id_acad=F.id_acad
GROUP BY F.Fauteuil
```

La jointure n'est nécessaire que pour connaître le nom de l'académicien : on aurait pu se contenter de son identifiant (et ne travailler qu'avec la table `Fauteuils`).

R 16.b La question qui reste est de savoir si le dernier académicien élu à un fauteuil est encore en vie ou non. On modifie la requête précédente pour faire apparaître la date de décès dans le résultat (ce qui rend la jointure nécessaire, car cette date n'apparaît que dans la table `Académiciens`).
On sélectionne alors les enregistrements pour lesquels l'attribut `DDD` n'est pas `NULL`.

```
SELECT Fauteuil, DDD AS "Date Vacance" FROM
 (SELECT F.Fauteuil AS Fauteuil, MAX(F.Election), A.Nom, A.DDD AS DDD
  FROM Fauteuils AS F
  JOIN Académiciens AS A ON A.id_acad=F.id_acad
  GROUP BY F.Fauteuil)
WHERE DDD IS NOT NULL
```

R 17. Les académiciens exclus

```
SELECT A.Nom, F.Election, F.Exclusion, A.DDD, F.Motif FROM Académiciens AS A
JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE F.Exclusion IS NOT NULL
ORDER BY F.Exclusion
```

Antoine Furetière fut exclu pour avoir publié son propre dictionnaire, agacé par la lenteur des travaux de l'Académie.

Les guerres mondiales

R 18.a On cherche les académiciens élus avant le 1er janvier 1918 et morts après cette date. Comme la date d'élection et la date de décès sont situées sur deux tables différentes, une jointure est nécessaire.

```
SELECT F.Fauteuil, A.Nom FROM Académiciens AS A
JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE A.DDD>"1918-01-01" AND F.Election<"1918-01-01"
ORDER BY F.Fauteuil
```

Maintenant qu'on connaît les fauteuils occupés, on peut en déduire ceux qui sont vacants : ce sont les autres !

```
SELECT Fauteuil FROM Fauteuils
WHERE Fauteuil NOT IN   -- table auxiliaire produite par une sous-requête
 (SELECT F.Fauteuil FROM Académiciens AS A
  JOIN Fauteuils AS F ON A.id_acad=F.id_acad
  WHERE A.DDD>"1918-01-01" AND F.Election<"1918-01-01")
GROUP BY Fauteuil       -- suppression des doublons
```

Pour que chaque fauteuil vacant n'apparaisse qu'une seule fois dans le résultat, on peut utiliser l'instruction `SELECT DISTINCT` [7.3] ou, comme ici, astucieusement regrouper les enregistrements selon l'attribut `Fauteuil`.

R 18.b Rien de bien différent : la requête est quasiment la même, le nombre de sièges occupés à peine plus faibles (la dernière élection remontait au 11 janvier 1940).

```
SELECT F.Fauteuil, A.Nom FROM Académiciens AS A
JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE A.DDD>"1944-10-12" AND F.Election<"1944-10-12"
ORDER BY F.Fauteuil
```

Doyens d'élection

R 19.a Au moyen d'une sous-requête, on tire de la table `Académiciens` une table auxiliaire, nommée `B`, qui contient les informations nécessaires à la suite du calcul : le nom, le sexe (un doyen pour chaque sexe) et l'identifiant (pour assurer la jointure avec la table `Fauteuils`) en sélectionnant bien entendu les académiciens vivants (puisqu'un doyen d'élection est, par définition, vivant).

```
SELECT Nom, MIN(F.Election) AS Election FROM
 (SELECT id_acad, Sexe, Nom FROM Académiciens
  WHERE DDD IS NULL) AS B
JOIN Fauteuils AS F ON B.id_acad=F.id_acad
GROUP BY B.Sexe
```

R 19.b Jusqu'en 1980, les académiciens étaient tous des hommes, ce qui va un peu simplifier la requête (inutile de prendre en compte l'attribut `Sexe`). On s'intéresse ici aux académiciens élus avant le 1er janvier 1675 et décédés après ce jour : il faut connaître leur nom et la date de leur élection.

```
SELECT A.Nom AS Nom, F.Election
FROM Académiciens AS A JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE A.DDD>"1675-01-01"  AND F.Election<"1675-01-01"
```

Remarque.— On constate que cette table contient 40 enregistrements : tous les fauteuils étaient occupés ce jour-là. À d'autres dates, on trouveraient moins d'enregistrements (cas de fauteuils vacants) et parfois plus d'enregistrements (cas de démission ou d'exclusion, avec un successeur élu avant la mort de l'ex-académicien). La requête suivante serait donc plus prudente : parmi les académiciens non décédés et pour chaque fauteuil, on retient le dernier académicien élu.

```
SELECT F.Fauteuil, A.Nom AS Nom, MAX(F.Election) AS Election
FROM Académiciens AS A JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE A.DDD > "1675-01-01"  AND F.Election<"1675-01-01"
GROUP BY F.Fauteuil
```

❧ Cette table contient toute l'information nécessaire pour calculer le doyen d'élection. Mais il ne s'agit pas seulement de trouver la daté d'élection du doyen, il s'agit d'identifier les deux doyens ! Le plus simple est alors de créer [17] une table auxiliaire avec le résultat de la requête précédente.

```
CREATE TABLE tab_aux AS
SELECT F.Fauteuil, F.id_acad, A.Nom AS Nom, MAX(F.Election) AS Election
FROM Académiciens AS A JOIN Fauteuils AS F ON A.id_acad=F.id_acad
WHERE A.DDD > "1675-01-01"  AND F.Election<"1675-01-01"
GROUP BY F.Fauteuil
```

Il reste à sélectionner dans la table auxiliaire les noms associés à la plus petite date d'élection, laquelle date est calculée au moyen d'une sous-requête.

```
SELECT Nom FROM tab_aux
WHERE Election=(SELECT MIN(Election) FROM tab_aux)
```

R 20. Les attributs `Fauteuil` et `Sexe` se trouvant dans deux tables distinctes, la jointure est nécessaire. La jointure étant faite, on compte le nombre d'enregistrements en les regroupant selon l'attribut `Fauteuil`, puis selon l'attribut `Sexe`.

```
SELECT F.Fauteuil, A.Sexe, COUNT(F.id_acad) AS Nbre
FROM Fauteuils AS F
JOIN Académiciens AS A ON A.id_acad=F.id_acad
GROUP BY F.Fauteuil, A.Sexe
```

En intervertissant les deux critères de regroupement, on ne modifierait que l'ordre des enregistrements dans le résultat.