T.C.d'I. — MPSI 1/MPSI 3/PC 1/PC* 1/MP 1/MP* 2

BASES DE DONNÉES

Dans tout le document, si $\mathscr{X}$ est un ensemble, on notera

$$\mathscr{X}^* = \bigcup_{n\geqslant 0} \mathscr{X}^n$$

l'ensemble des uplets dont les composantes sont dans $\mathscr{X}$. Si $x = (x_0\,;\, x_1\,;\, \ldots\,;\, x_{m-1})$, à composantes pas forcément dans le même ensemble, on notera $|x|$ sa longueur (c'est-à-dire m, son nombre de composantes) et si $y = (y_0\,;\, y_1\,;\, \ldots\,;\, y_{n-1})$ est un autre uplet, on notera

$$xy = (x_0\,;\, \ldots\,;\, x_{m-1}\,;\, y_0\,;\, \ldots\,;\, y_{n-1})$$

le résultat de sa concaténation avec x. Le nulluplet (l'uplet à zéro composante) sera noté ε ; c'est un élément neutre à gauche et à droite pour la concaténation.

Rappelons la définition du *produit cartésien* : si $\mathscr{E}$ et $\mathscr{E}'$ sont des ensembles d'uplets, on note $\mathscr{E}\times\mathscr{E}'$ l'ensemble de tous les concaténés d'un élément de $\mathscr{E}$ avec un élément de $\mathscr{E}'$, et plus généralement si $(\mathscr{E}_i)_{i\in \mathrm{I}}$ est une famille d'ensembles d'uplets, avec un ensemble d'indices I totalement ordonné, on note

$$\prod_{i\in \mathrm{I}} \mathscr{E}_i$$

l'ensemble de tous les uplets obtenus en concaténant, dans l'ordre de I, des éléments pris dans chacuns des ensembles $\mathscr{E}_i$.

Avec cette définition légèrement retouchée du produit cartésien, on identifie donc « un couple formé d'un couple et d'un triplet » avec « un quintuplet ».

§1. Définitions

Voici une *table* : elle s'appelle `Naissances` et elle liste les prénoms donnés lors des naissances en France.

`Naissances`					
`id`	`genre`	`prénom`	`jour`	`mois`	`année`
1	F	Lilith	29	février	2000
2	G	Adam	1	mars	2000
3	F	Ève	2	mars	2000
4	...	...	...	...	...

Cette table possède six *colonnes* : `id` (à valeurs dans $\mathbf{N}$), `genre` (une chaîne de caractères), `prénom` (une chaîne de caractères), `jour` (à valeurs dans $\mathbf{N}$) `mois` (une chaîne de caractères) et `année` (à valeurs dans $\mathbf{N}$).

Une *base de données* $(\mathscr{A}, \mathscr{T})$ est la donnée

i) d'un ensemble $\mathscr{A}$ d'*attributs*,

ii) pour chaque attribut $\mathrm{A} \in \mathscr{A}$ d'un ensemble $\mathrm{dom}(\mathrm{A})$ appelé le *domaine* de l'attribut,

iii) d'un ensemble $\mathscr{T}$ de *tables*.

Une *table* T est la donnée

i) d'un ensemble $\mathscr{S}(\mathrm{T}) \in \mathscr{A}^*$ de *colonnes*, choisies parmi les attributs de la base de données à laquelle appartient cette table. On dit que $\mathscr{S}(\mathrm{T})$ est le *schéma relationnel* de la table,

ii) d'un ensemble de *lignes* (on dit aussi les *entrées*) $\mathscr{E}(\mathrm{T}) \subseteq \prod_{\mathrm{A} \in \mathscr{S}(\mathrm{T})} \mathrm{dom}(\mathrm{A})$. Par commodité on notera simplement $\mathrm{dom}(\mathscr{S}(\mathrm{T}))$ ce dernier produit cartésien.

Sélections. À partir d'une table, on peut en construire d'autres par diverses opérations. La première est la *sélection*. Soit T une table de schéma relationnel $\mathscr{S}$, et soit $\mathscr{P}$ une assertion portant sur les éléments de $\mathrm{dom}(\mathscr{S})$. On note

$$\sigma_{\mathscr{P}}(\mathrm{T}) = \{x \in \mathscr{E}(\mathrm{T}) \mid \mathscr{P}(x)\}$$

la table ayant le même schéma relationnel que T et dont les lignes sont le sous-ensemble de celles de T qui vérifient $\mathscr{P}$. Une sélection consiste donc à retirer certaines des lignes de la table.

Par exemple, la table répertoriant les naissances des filles se note

$$\sigma_{\texttt{genre}=\mathrm{F}}(\texttt{Naissances}).$$

En langage SQL, cette *même* opération s'écrit

```
SELECT *
 FROM  Naissances
 WHERE genre = "F"
```

et c'est le mot-clé `WHERE` qui indique la sélection.

Projections. La *projection* est une opération qui consiste à supprimer une partie des colonnes d'une table (on peut aussi s'en servir pour dupliquer certaines colonnes, ou pour en changer l'ordre). Soit T une table de schéma relationnel $\mathscr{S}$, et soit $\mathscr{S}'$ un uplet d'attributs dont toutes les composantes apparaissent dans $\mathscr{S}$. On note

$$\pi_{\mathscr{S}'}(\mathrm{T}) = \{(x_{\mathrm{A}'})_{\mathrm{A}' \in \mathscr{S}'} \mid (x_{\mathrm{A}})_{\mathrm{A} \in \mathscr{S}} \in \mathscr{E}(\mathrm{T})\}\,.$$

Par exemple, si l'on veut ne conserver que les identifiants, les prénoms et les années, on écrira

$$\pi_{\texttt{id,prénom,année}}(\texttt{Naissance})$$

pour obtenir la table

id	prénom	année
1	Lilith	2000
2	Adam	2000
3	Ève	2000
...	...	...

En langage SQL, cette même opération s'écrit

```
SELECT id, prénom, année
 FROM Naissances
```

et c'est le mot-clé `SELECT` qui indique la projection. On peut bien sûr réaliser en même temps une sélection et une projection, par exemple

$$\pi_{\texttt{id,prénom,année}}(\sigma_{\texttt{genre}=\texttt{F}}(\texttt{Naissances}))$$

s'écrira en SQL

```
SELECT id, prénom, année
 FROM  Naissances
 WHERE genre = "F"
```

Le mot-clé `FROM` indique la table dans laquelle se fait la sélection/la projection. Le caractère `*` en face de l'instruction `SELECT` indique l'absence de projection (on conserve toutes les colonnes).

Renommage. Soit $\mathscr{S}$ un uplet d'attributs, soit A une composante de $\mathscr{S}$, et soit B un autre attribut tel que dom(A) $\subseteq$ dom(B). Si T est une table de schéma relationnel $\mathscr{S}$, et si $\mathscr{S}'$ est l'uplet obtenu en remplaçant A par B, alors on note

$$\rho_{\text{A}\to\text{B}}(\text{T}) \quad \text{ou} \quad \pi_{\ldots,\text{A}\to\text{B},\ldots}(\text{T})$$

la table de schéma relationnel $\mathscr{S}'$, ayant les mêmes lignes que T, mais dans laquelle on a changé le nom de la colonne A en B (ci-dessus, on met à la place des pointillés toutes les composantes n'ayant pas changé de nom). On peut évidemment faire plusieurs renommages d'un seul coup.

Par exemple, dans la table des naissances, si on veut renommer la première colonne, on écrira

$$\pi_{\texttt{id}\to\texttt{numéro,genre,prénom,jour,mois,année}}(\texttt{Naissances})$$

tandis qu'en langage SQL la même opération se note

```
SELECT id AS numéro, genre, prénom, jour, mois, année
 FROM Naissances
```

et c'est c'est le mot-clé `AS` qui indique le renommage.

Exercice 1 — Écrire dans l'algèbre relationnelle (c'est-à-dire à l'aide des opérateurs σ, π, etc.) la requête qui donne la table regroupant les naissances des filles. Puis la traduire en langage SQL.

Exercice 2 — Comment obtenir toutes les naissances du 29 février ?

Exercice 3 — Écrire dans l'algèbre relationnelle puis en SQL la requête qui dresse la liste des années bissextiles.

Exercice 4 — Écrire une requête qui donne la liste des naissances hivernales.

§2. Clés

Soit T une table de schéma relationnel $\mathscr{S}$. Soit $\mathscr{C}$ un uplet dont les composantes sont prises dans $\mathscr{S}$: on dit que $\mathscr{C}$ est une *clé* de T. On dit que $\mathscr{C}$ est une *clé primaire* si la fonction $\pi_{\mathscr{C}}$ est injective sur $\mathscr{E}$(T), autrement dit si les composantes de $\mathscr{C}$ identifient *de manière unique* les lignes de T.

Par exemple, dans la table `Naissances`, le quadruplet (`prénom`, `jour`, `mois`, `année`) n'est sans doute pas une clé primaire, car il est fort possible que plusieurs Ève soient nées le 2 mars 2000. D'une manière générale, il est difficile de donner une clé primaire : celà demande de connaître *toutes* les lignes de la table. En particulier, une clé qu'on annonce primaire peut ne plus l'être si l'on ajoute des lignes à la table...

Pour remédier à ce problème, on ajoute en générale une colonne `id` (comme « identificateur ») qui est *auto-incrémentée* : à chaque fois qu'une ligne est ajoutée à la table, sa valeur est augmentée de 1, ce qui garantit que le singulet $\mathscr{C}$ = (`id`) est une clé primaire. C'est le créateur de la table qui indique aux utilisateurs que cette colonne constitue une clé primaire.

Exercice 5 — Pourquoi la Sécurité Sociale attribue-t-elle des numéros de sécurité de sociale, pourquoi le FISC attribue-t-il des numéros fiscaux, pourquoi les universités attribuent-elles des numéros d'étudiants et pourquoi l'Éducation Nationale attribue-t-elle des INE ?

Exercice 6 — Qu'est-ce qui sert de clé primaire pour les voitures circulant en France ?

Exercice 7 — Qu'est-ce que les codes AITA ? Et que sont CDG et JFK ?

§3. Opérations ensemblistes

Si T et T′ sont deux tables ayant le même schéma relationnel $\mathscr{S}$, on peut créer les nouvelles tables $\mathrm{T} \cup \mathrm{T}'$, $\mathrm{T} \cap \mathrm{T}'$ et $\mathrm{T} - \mathrm{T}'$, toutes de schéma $\mathscr{S}$ également, et dont les lignes sont respectivement $\mathscr{E}(\mathrm{T}) \cup \mathscr{E}(\mathrm{T}')$, $\mathscr{E}(\mathrm{T}) \cap \mathscr{E}(\mathrm{T}')$ et $\mathscr{E}(\mathrm{T}) - \mathscr{E}(\mathrm{T}')$. En langage SQL, ces trois opérations se notent `UNION`, `INTERSECT` et `EXCEPT` (ou `MINUS`).

Dans une base de données à une seule table, ces opérations peuvent toujours êtres remplacées par des opérations booléennes dans un critère de sélection. Ainsi par exemple, si l'on veut tous les Adam *sauf* ceux nés en 2000, on peut écrire

$$\sigma_{\texttt{prénom}=\mathrm{Adam},\texttt{année}>2000}(\texttt{Naissances}) \cup \sigma_{\texttt{prénom}=\mathrm{Adam},\texttt{année}<2000}(\texttt{Naissances})$$

ou bien

$$\sigma_{\texttt{prénom}=\mathrm{Adam},(\texttt{année}<2000 \vee \texttt{année}>2000)}(\texttt{Naissances}),$$

ce qui correspond en langage SQL respectivement à

```
(SELECT *
  FROM  Naissances
  WHERE prénom = "Adam" AND année > 2000)
UNION
(SELECT *
  FROM  Naissances
  WHERE prénom = "Adam" AND année < 2000)
```

ou

```
SELECT *
 FROM  Naissances
 WHERE prénom = "Adam" AND (année > 2000 OR année < 2000)
```

On peut bien sûr aussi utiliser l'opérateur « différent de » qui en SQL se note indifféremment `<>` ou `!=`.

Exercice 8 — On dispose d'une seconde table `Naissances_RU` qui répertorie les naissances au Royaume-Uni. Comment obtenir tous les prénoms donnés dans les deux pays ? Et le complémentaire, c'est-à-dire ceux qui ne sont donnés que d'un seul côté de la Manche ?

Exercice 9 — Comment obtenir la liste de tous les prénoms mixtes ?

§4. Produits cartésiens, jointures

On a jusqu'à présent travaillé avec une base de données à une seule table. On va maintenant en utiliser plusieurs.

Une base de données `France` contient deux tables. La première s'appelle `Villes` et possède quatre colonnes : `id` (un entier naturel), `nom` (une chaîne de caractères), `population` (un entier naturel) et `département` (un entier).

Villes			
id	nom	population	département
1	Paris	2 187 526	75
2	Lyon	516 092	69
3	...	...	...

La deuxième s'appelle `Départements` et possède quatre colonnes : `numéro` (une chaîne de caractères), `nom` (une chaîne de caractères) et `préfecture` (un entier naturel).

Départements		
numéro	nom	préfecture
01	Ain	18902
69	Rhône	2
...	...	...

Si l'on a deux tables T_1 et T_2, de schémas relationnels respectifs $\mathscr{S}_1$ et $\mathscr{S}_2$, on note $\mathrm{T}_1 \times \mathrm{T}_2$ la table de schéma $\mathscr{S}_1\mathscr{S}_2$ et dont les entrées sont

$$\mathscr{E}(\mathrm{T}_1 \times \mathrm{T}_2) = \{x_1x_2 \mid x_1 \in \mathscr{E}(\mathrm{T}_1),\, x_2 \in \mathscr{E}(\mathrm{T}_2)\}\,.$$

Si l'on utilise l'exemple ci-dessus, la table `Villes` × `Départements` s'obtient ainsi en SQL.

```
SELECT *
 FROM Villes JOIN Départements
```

Sauf que cette table a très peu d'intérêt : on y trouve par exemple les lignes suivantes.

id	nom	population	département	numéro	nom	préfecture
1	Paris	2 187 526	75	69	Rhône	2
2	Lyon	516 092	69	69	Rhône	2
...	...	...	...	...	...	...

Si la deuxième ligne nous intéresse (elle permet de savoir que Lyon est situé dans le Rhône), la première ne sert à rien (Paris n'est pas dans le Rhône, même si cette ligne est bien dans le produit cartésien).

On souhaite donc ne conserver que les lignes pour lesquelles les valeurs des colonnes `département` et `numéro` coïncident. L'opération correspondante s'appelle une *jointure symétrique*, dans cet exemple elle se note

$$\texttt{Villes} \underset{\texttt{département=numéro}}{\bowtie} \texttt{Départements}$$

et en langage SQL on l'écrit ainsi.

```
SELECT *
 FROM Villes JOIN Départements
       ON département = numéro
```

Améliorons encore un peu cet exemple : il ne sert à rien de conserver les deux colonnes identiques, on va donc faire une projection. Et en profiter pour renommer les deux colonnes `nom` pour qu'elle n'aient plus le même. On écrit comme ceci

$$\pi_{\texttt{id,Villes.nom}\to\texttt{vnom,...,Département.nom}\to\texttt{dnom,...}}\left(\texttt{Villes} \underset{\texttt{département=numéro}}{\bowtie} \texttt{Départements}\right)$$

ou comme cela.

```
SELECT id, Villes.nom AS vnom, population, département,
       Département.nom as dnom, préfecture
 FROM Villes JOIN Départements
       ON département = numéro
```

On remarquera la syntaxe, lorsqu'il y *ambiguité* dans le nom des colonnes (deux colonnes de deux tables différentes ont le même nom) : on fait précéder le nom de ce colonnes par un point (c'est un *opérateur de résolution de portée*) et le nom de la table dont est issue la colonne.

Exercice 10 — Comment obtenir le numéro de la Creuse ? Et le nom du département 972 ?

Exercice 11 — Écrire dans l'algèbre relationnelle puis en SQL la requête qui donne le nom de la préfecture de la Haute-Loire.

Exercice 12 — Comment obtenir toutes les communes ultramarines ?

Exercice 13 — Écrire dans l'algèbre relationnelle puis en SQL la requête qui donne le nom et la population de toutes les villes de l'Ariège.

Exercice 14 — Comment obtenir les noms de toutes les préfectures, avec leurs populations ?

Exercice 15 — Quelle requête permet d'obtenir tous les noms des départements dont au moins ville s'appelle Saint-Martin ?

Exercice 16 — Écrire dans l'algèbre relationnelle puis en SQL la requête qui dresse la liste de toutes les communes mortes pour la France (non reconstruites, et non rattachée à une autre commune ; il en reste six aujourd'hui).

§5. Agrégations

Une *fonction d'agrégation* sur l'ensemble $\mathscr{X}$ est une fonction dont le domaine de définition est $\mathscr{X}^*$ (auquel il faudra parfois retirer le nulluplet). Il y en a cinq à connaître :

i) la fonction de *comptage*, qui renvoie le nombre de composantes ; elle a du sens pour n'importe quel $\mathscr{X}$ et est à valeurs dans $\mathbf{N}$. En SQL elle se note `COUNT`,

ii) la *moyenne*, qui a du sens lorsque $\mathscr{X}$ est une partie d'un ensemble de nombres $\mathbf{K}$, elle est définie sur $\mathscr{X}^* - \{\varepsilon\}$ et est à valeurs dans $\mathbf{K}$. En SQL elle se note `AVG`,

iii) le *maximum* (tout pareil) qui en SQL se note `MAX`,

iv) le *minimum* (tout pareil aussi) qui en SQL se note `MIN`,

v) la *somme*, qui a du sens lorsque $\mathscr{X}$ est une partie d'un ensemble de nombres $\mathbf{K}$, elle est définie sur $\mathscr{X}^*$ tout entier (rappelons qu'une somme vide vaut zéro) et est à valeurs dans $\mathbf{K}$. En SQL elle se note `SUM`.

Ceci étant vu, l'*agrégation* est une opération qui consiste à regrouper en paquets les lignes d'une table. Plus précisément, soit T une table, de schéma relationnel $\mathscr{S}$. Soit $\mathscr{S}'$ un uplet dont les composantes sont prises parmi celles de $\mathscr{S}$. On note $\gamma_{\mathscr{S}'}(\mathrm{T})$ la table de schéma relationnel $\mathscr{S}'$ et dont les lignes sont

$$\mathscr{E}(\gamma_{\mathscr{S}'}(\mathrm{T})) = \{(x_{\mathrm{A}'})_{\mathrm{A}'\in\mathscr{S}'} \mid (x_{\mathrm{A}})_{\mathrm{A}\in\mathscr{S}} \in \mathscr{E}(\mathrm{T})\}\,.$$

Dit comme cela, on ne voit pas la différence avec une projection. La subtilité vient du fait que *pendant* le regroupement en paquets, on peut appliquer (à chaque paquet) une fonction d'agrégation. Par exemple

$$\gamma_{\texttt{genre},\mathrm{card}(\texttt{id})}(\mathrm{Naissances})$$

qui se note en SQL

```
SELECT genre, COUNT(id)
 FROM     Naissances
 GROUP BY genre
```

donne le nombre de filles et de garçons dans la table. Et voici le résultat.

genre	
F	14 172 378
G	13 998 081

Le 14 172 378 a été obtenu en appliquant la fonction de comptage aux valeurs de la colonne `id` dans tout le paquet des entrées dont la colonne `genre` vaut F, et la valeur 13 998 081 en faisant la même chose dans tout le paquet des entrées dont cette même colonne `genre` vaut G.

Au passage, on remarque que la deuxième colonne de la table obtenue n'a pas de nom (c'est le résultat d'un calcul), on peut lui en donner un avec l'opérateur de renommage. En fait, si l'on ré-exploite cette colonne par la suite, on *doit*. On écrira donc

$$\gamma_{\texttt{genre},\mathrm{card}(\texttt{id})\to\texttt{nb}}(\texttt{Naissances})$$

ou en SQL

```
SELECT genre, COUNT(id) AS nb
 FROM     Naissances
 GROUP BY genre
```

pour obtenir la table suivante.

genre	nb
F	14 172 378
G	13 998 081

On aura remarqué que c'est le mot-clé `GROUP BY` qui indique l'agrégation (suivi des critères de formation des paquets).

Sélections en amont, sélections en aval. Voyons un exemple plus compliqué : une base de données `Construction` contient deux tables. La première s'appelle `Pièces` et possède trois colonnes : `id` (un entier naturel), `coût` (un nombre) représentant le coût (en euros) de production unitaire, et `encombrement` (un nombre) indiquant l'espace moyen (en centimètres cubes) occupé par la pièce dans une boîte.

Pièces		
id	coût	encombrement
1	0,125	3,2
2	0,008	1,4
3	...	...

La deuxième s'appelle `Assemblages` et possède quatre colonnes : `modèle` (un entier naturel), `pièce` (un entier naturel) et `quantité` (un entier naturel). Chaque ligne indique le nombre d'exemplaires d'une pièce donnée dans un modèle donné.

Assemblages		
modèle	pièce	quantité
6301	18	2
...	...	...

Cherchons le nombre de « pièces uniques » utilisées dans chaque modèle, c'est-à-dire une pièce qui n'est utilisée qu'en un seul exemplaire dans le modèle.

On commence par effectuer une sélection pour ne conserver dans la table que les pièces uniques, puis on effectue le regroupement par modèle :

$$\gamma_{\texttt{modèle},\text{card}(*)\to\texttt{nb}}(\sigma_{\texttt{quantité}=1}(\texttt{Assemblages})).$$

En SQL on écrit ceci.

```
SELECT modèle, COUNT(*) AS nb
 FROM     Assemblages
 WHERE    quantité = 1
 GROUP BY modèle
```

On parle de sélection *en amont* car elle est effectuée *avant* le regroupement par paquets, c'est (on l'a déjà vu) le mot-clé `WHERE` qui l'indique et il est toujours situé *avant* le `GROUP BY`.

Cherchons maintenant les modèles qui utilisent plus de mille pièces. Le nombre de pièces par modèle s'obtient avec l'agrégation $\gamma_{\texttt{modèle},\text{somme}(\texttt{quantité})\to\texttt{nb}}(\texttt{Assemblages})$. Pour ne conserver que les modèles à plus de mille pièces, on rajoute donc une sélection :

$$\sigma_{\texttt{nb}>1000}(\gamma_{\texttt{modèle},\text{somme}(\texttt{quantité})\to\texttt{nb}}(\texttt{Assemblages})).$$

En langage SQL on écrit cela.

```
SELECT modèle, SUM(quantité) AS nb
 FROM     Assemblages
 GROUP BY modèle
 HAVING   nb > 1000
```

On parle de sélection *en aval*, parce qu'elle est effectuée *après* le regroupement par paquets (d'ailleurs, l'attribut `nb` n'a pas de sens tant que les paquets ne sont pas faits). C'est le mot-clé `HAVING` qui indique une sélection en aval, et il est toujours situé *après* le `GROUP BY`.

Exercice 17 — Quelle requête permet d'obtenir le nombre de pièces distinctes par modèle ?

Exercice 18 — Écrire la requête SQL qui calcule le coût total de chaque modèle, puis celle qui calcule l'encombrement total. Enfin, écrire une requête qui calcule pour chaque modèle le rapport `coût/encombrement`.

Exercice 19 — Une troisième table `Achats` recense les achats de modèles. Elle possède deux colonnes `modèle`, `date`. Écrire la requête qui calcule le nombre d'exemplaire de chaque pièce qu'il a fallu produire en 2018 pour satisfaire tous les achats de cette année.

Exercice 20 — Écrire une requête, dans l'algèbre relationnelle puis en SQL, qui détermine la population de la Haute-Saône.

Exercice 21 — Comment obtenir toutes les préfectures de plus de 100 000 habitants ?

Exercice 22 — Écrire une requête qui dresse la liste de tous les départements ayant au moins deux villes de plus de 100 000 habitants.

Exercice 23 — Comment obtenir le nombre de naissances de filles en 2000 ?

§6. Requêtes imbriquées

Il est facile de composer des opérations sur les tables : on peut donner un nom à une table résultat d'une certaine opération, puis ré-utiliser ce nom dans une autre opération.

En SQL, c'est moins commode : il n'y a pas d'opérateur d'affectation, donc on ne peut pas utiliser le résultat d'une requête précédente. Il faut recopier cette requête à l'intérieur d'une requête plus grosse. Voyons par exemple comment déterminer s'il existe un modèle qui n'utilise aucune pièce en double. On peut écrire

```
SELECT MIN(m)
 FROM (SELECT modèle, MAX(quantité) AS m
        FROM     Assemblages
        GROUP BY modèle)
```

qui répond à la question ; voyons comment rédiger cela (ou un exemple plus compliqué encore) sur une copie.

On commence par écrire une sous-requête qui donne, pour chaque modèle, le nombre maximal d'utilisation d'une même pièce.

```
SELECT modèle, MAX(quantité) AS m
 FROM     Assemblage
 GROUP BY modèle
```

On insère cette sous-requête (qu'on désignera `SR_1`) dans une deuxième requête qui calcule la valeur minimale de m.

```
SELECT MIN(m)
 FROM (SR_1)
```

Remarque : on ne peut pas donner de nom au résultat de la requête principale, mais dans le même temps, *on est obligé de nommer*, avec l'opérateur `AS`, les tables utilisées comme opérandes d'une jointure.

Exercice 24 — Écrire une requête SQL qui dresse la liste de tous les départements dont la préfecture concentre au moins 10 % de la population.

Exercice 25 — Comment obtenir la moins peuplée parmi les villes les plus peuplées de chaque département ? Et la ville la plus peuplée parmi les villes les moins peuplées de chaque département ?

Exercice 26 — Écrire une requête qui détermine s'il existe deux départements distincts dont la somme des habitants vaut exactement trois millions. Puis recommencer avec trois départements.

Exercice 27 — Écrire une requête qui détermine tous les couples de modèles (m, m') tels qu'en réunissant les pièces de m et de m', on puisse construire le modèle 6450.

Exercice 28 — Comment vérifier s'il y au moins un jour où ne sont nées, en France, que des filles ?

§7. Problèmes d'appartenance

Une base de données `CDI` contient trois tables. La première s'appelle `Ouvrages` et possède trois colonnes : `id` (un entier naturel), `titre` (une chaîne de caractères), `auteur` (une chaîne de caractères).

Ouvrages		
id	titre	auteur
1	Le rivage des Syrthes	Gracq, Julien
2	Belle du Seigneur	Cohen, Albert
3	...	...

La deuxième s'appelle **Élèves** et possède quatre colonnes : `id` (un entier naturel), `nom`, `prénom`, `classe` (toutes les trois des chaînes de caractères).

Élèves			
id	nom	prénom	classe
1	Lacouverthur--Meugrate	Sandra	Term 1
2	Méjust	Sévère	Term 2
3	...	...	...

La troisième s'appelle **Opérations** et possède quatre colonnes : `livre` (un entier naturel), `élève` (un entier naturel), `opération` (une chaîne de caractères, égale à « sortie » (lorsqu'on emprunte un livre) ou à « entrée » (lorsqu'on le rapporte) et `date`. On ajoute une nouvelle ligne à cette table à chaque fois qu'un livre est emprunté ou rendu, avec les identificateurs du livre et de l'emprunteur et la date de l'action.

Opérations			
livre	élève	opération	date
1	481	sortie	01/09/2019 – 13:45:02
1	481	entrée	04/09/2019 – 07:57:15
...	...	...	...

Exemple 1. Écrivons une requête qui teste si *Le rouge et le noir*, de Stendhal, existe dans ce CDI.

```
SELECT *
 FROM  Ouvrages
 WHERE titre = "Le rouge et le noir" AND auteur = "Stendhal"
```

S'il y a des résultats, c'est que le livre existe.

Exemple 2. Écrivons une requête qui détermine la liste de tous les exemplaires n'ayant jamais été empruntés. On va utiliser la différence ensembliste, en enlevant de la liste des exemplaires ceux qui apparaissent dans au moins une opération.

```
(SELECT id AS x
 FROM Ouvrages)
EXCEPT
(SELECT livre
 FROM    Opérations
 GROUP BY livre)
```

Si on veut les titres correspondants, on insère cette sous-requête (que nous désignerons `SR_1`) dans une deuxième requête.

```
SELECT id, titre, auteur
 FROM Ouvrages JOIN (SR_1) AS t
       ON x = livre
```

Exercice 29 — Écrire une requête qui dresse la liste des titres (et non pas des exemplaires) n'ayant jamais été empruntés.

Exercice 30 — Comment obtenir la liste des élèves ayant en leur possession un livre du CDI ?

Exercice 31 — Écrire une requête qui détermine s'il est possible d'emprunter *L'homme qui rit* de Victor Hugo.

Exercice 32 — Quelle requête permet de savoir s'il existe des classes dans lesquelles aucun élève n'a jamais emprunté quoi que ce soit au CDI ?

Exercice 33 — Écrire une requête qui dresse la liste des livres pas encore rendus le 21 juin 2019 au soir ; puis une autre dressant la liste des élèves à contacter pour leur demander de les rapporter.

Exercice 34 — Écrire une requête qui dresse la liste des modèles qu'on peut construire à partir des pièces du seul modèle 7544.

Exercice 35 — Écrire une requête qui dresse la liste de tous les couples de modèles n'utilisant aucune pièce en commun.

Exercice 36 — Proposer une requête qui renvoie les noms des départements français n'ayant aucune commune d'exactement mille habitants.

Exercice 37 — Comment savoir s'il y a en Haute-Savoie deux communes ayant exactement le même nombre d'habitants ?

Exercice 38 — Proposer une requête qui permet de vérifier qu'il y a eu au moins une naissance par jour depuis la mise en place de la base de données répertoriant les naissances en France.

§8. Problèmes de rangs

Une base de données `BAC_2019` regroupe les notes des candidats au baccalauréat. Elle est constituée de deux tables. La première s'appelle `Élèves` est possède sept colonnes : `INE` (une chaîne de caractères, clé primaire), `nom`, `prénom`, `lycée`, `ville`, `académie` (des chaînes de caractères) et `ddn` (une date) représentant la date de naissance du candidat.

Élèves						
INE	nom	prénom	lycée	ville	académie	ddn
2318099201Z	Honnête	Camille	Jean Favard	Guéret	Limoges	30/08/2001
1194007289S	Manvussa	Gérard	Eugène Jamot	Aubusson	Limoges	03/01/2001
...	...	...	...	...	...	...

La deuxième table s'appelle `Notes` et possède quatre colonnes : `élève` (une chaîne, correspondant à l'INE), `note` et `coef` (des nombres) et `matière` (une chaîne de caractères).

Notes			
élève	note	coef	matière
1380901923R	12,5	3	LV1
1379882977T	07,0	6	mathématiques
...	...	...	...

Plus grand élément. Écrivons une requête qui donne l'académie ayant le plus de candidats. Première option : on fait une agrégation, et on ordonne les résultats avec le mot-clé `ORDER BY` (et `DESC` pour avoir l'ordre décroissant).

```
SELECT académie, COUNT(*) AS nb
 FROM     Élèves
 GROUP BY académie
 ORDER BY nb DESC
```

La première ligne de la table qui résulte de cette requête est la réponse, et on peut même voir s'il y a des académies ex-æquo.

Deuxième option, en se passant de `ORDER BY`. On commence de la même manière.

```
SELECT académie, COUNT(*) AS nb
 FROM     Élèves
 GROUP BY académie
```

Puis on insère cette sous-requête (qu'on désignera `SR_1`) dans une deuxième requête qui calcule la valeur maximale de `nb`.

```
SELECT MAX(nb) AS m
 FROM (SR_1)
```

Enfin, on insère cette deuxième sous-requête (qu'on désignera `SR_2`) dans une troisième requête qui trouve toutes les académies réalisant le maximum.

```
SELECT académie, nb
 FROM  (SR_1) AS t1 JOIN (SR_2) AS t2
 WHERE nb = m
```

On verra plusieurs lignes dans le résultat s'il y a des académies ex-æquo.

Deuxième plus grand. Venons-en au problème plus subtil de la deuxième académie ayant le plus de candidats. Première observation : il n'y a pas toujours de *deuxième* : s'il y a deux éléments premiers ex-æquo, le suivant est directement *troisième*.

Seconde observation : la solution avec le `ORDER BY` fonctionne toujours, sans rien changer (on lit la deuxième ligne plutôt que la première). Elle permet de repérer la situation d'ex-æquo. Elle est très simple, et c'est vraissemblablement ce qu'un utilisateur ferait si on lui demandait d'interroger la base de données.

En revanche pour avoir (d'une manière général) le n-ième *n'écrit pas*

```
SELECT académie, COUNT(*) AS nb
 FROM     Élèves
 GROUP BY académie
 LIMIT    1
 OFFSET   n
```

(le mot-clé `LIMIT` restreignant la table des résultats à un certains nombres de lignes, et le mot-clé `OFFSET` indiquant de « sauter » un certain nombre de lignes au départ) parce que le résultat serait *faux* dans la situation où il y a des valeurs ex-æquo.

Réfléchissons maintenant dans l'optique des écrits : que fait-on si le sujet interdit d'utiliser l'instruction `GROUP BY` ? Réutilisons les sous-requêtes `SR_1` `SR_2` ci-dessus pour éliminer *toutes* les académies réalisant le plus grand nombre de candidats.

```
SELECT académie, nb
 FROM  (SR_1) AS t1 JOIN (SR_2) AS t2
 WHERE nb < m
```

Désignons par `SR_3` cette sous-requête et insérons-la dans une quatrième requête qui donne la deuxième plus grande valeur.

```
SELECT MAX(nb) AS sous_max
 FROM (SR_3)
```

Enfin insérons cette sous-requête (qu'on désignera `SR_4`) dans une autre qui détermine toutes les académies réalisant cette valeur `sous_max`.

```
SELECT académie, nb
 FROM  (SR_3) AS t1 JOIN (SR_4) AS t2
 WHERE nb = sous_max
```

Insistons : en cas d'égalité, on voit plusieurs résultats ; mais surtout s'il y avait plusieurs premiers ex-æquo alors ceci ne correspond pas à la définition habituelle du deuxième (aux jeux olympiques, si les deux premiers arrivent ensemble, alors ils ont tous les deux une médaille d'or et la médaille d'argent n'est pas attribuée).

Autre remarque (est-elle nécessaire ?) : cette solution, qui donne un sens ambigu au mot « deuxième », est bien plus compliquée que la solution avec `ORDER BY`.

Exercice 39 — Comment obtenir le prénom féminin le plus attribué (en France) en 2000 ?

Exercice 40 — Comment connaître le premier prénom féminin à avoir été attribué plus de mille fois dans une même année ? Et plus de dix fois dans un même jour ?

Exercice 41 — Écrire une requête SQL qui détermine le département le plus peuplé.

Exercice 42 — Écrire une requête SQL qui détermine la ville la plus peuplée de chaque département.

Exercice 43 — Écrire une requête qui détermine les départements dont la préfecture n'est pas la ville la plus peuplée.

Exercice 44 — Comment obtenir le prix du modèle ayant le plus de pièces ? Et le nombre de pièces du modèle le plus cher ?

Exercice 45 — Écrire une requête qui dresse la liste des modèles utilisant au moins deux pièces parmi les dix plus chères.

Exercice 46 — Écrire une requête qui dresse la liste des modèles dont au moins un quart des pièces font partie des 5 % plus petites parmi toutes les pièces existantes.

Exercice 47 — Comment obtenir le titre ayant été le plus emprunté au CDI ?

Exercice 48 — Écrire une requête qui permet de déterminer l'élève ayant emprunté le plus de livres, puis celle qui permet de déterminer la classe ayant emprunté le plus de livres.

Exercice 49 — Écrire une requête qui renvoie le titre ayant été emprunté le plus de fois par un même élève.

Exercice 50 — Comment savoir si Jean Frémy, du lycée Paul Painlevé à Oyonnax, a eu son baccalauréat ?

Exercice 51 — Écrire une requête qui donne le pourcentage de réussite au bac par académie.

Solutions de quelques exercices

3) On part du principe qu'il y au moins une naissance par jour, ou en tout cas qu'il y en a eu au moins une chaque 29 février. On peut alors écrire $\pi_{\texttt{année}}(\sigma_{\texttt{jour}=29 \wedge \texttt{mois}=\text{février}}(\texttt{Naissances}))$, ou bien la même chose en SQL.

```
SELECT année
 FROM  Naissances
 WHERE jour = 29 AND mois = "février"
```

4) Pour simplifier on va dire que l'hiver commence le 22 décembre et se termine le 19 mars.

```
SELECT *
 FROM  Naissances
 WHERE (mois = "décembre" AND jour >= 22)
    OR (mois = "janvier")
    OR (mois = "février")
    OR (mois = "mars" AND jour < 20)
```

9) On fait l'intersection de deux tables : celle listant les prénoms féminins, et celle listant les prénoms masculins.

```
(SELECT prénom
  FROM  Naissances
  WHERE genre = "F")
INTERSECT
(SELECT prénom
  FROM  Naissances
  WHERE genre = "G")
```

12) Les départements de métropole ont des numéros au plus égaux à 95. On obtient donc les communes d'Outre-mer avec la requête suivante.

```
SELECT nom
 FROM  Villes
 WHERE département > 95
```

À noter que dans cette base de donnée on a simplifié un peu le problème, en « oubliant » que les départements de Corse n'ont pas des numéros entiers : ce sont 2A et 2B. Si l'on représente les codes départementaux par des chaînes, pour contourner ce problème, la requête précédente se complique un peu.

```
SELECT nom
 FROM  Villes
```

```
 WHERE département != "2A"
   AND département != "2B"
   AND CONVERT(INT, département) > 95
```

14)

```
SELECT Villes.nom, population
 FROM  Villes JOIN Départements
        ON département = numéro
 WHERE id = préfecture
```

15) Ici notons bien qu'on veut juste les *noms* des départements concernés.

```
SELECT Département.nom
 FROM Villes JOIN Départements
       ON département = numéro
 WHERE Villes.nom = "Saint-Martin"
```

16) Ces communes sont caractérisées par le fait d'avoir une population nulle (leur maire, en particulier, est désigné parmi les habitants d'une commune voisine). Elles sont toutes situées dans la Meuse.

```
SELECT Villes.nom, Départements.nom
 FROM Villes JOIN Départements
       ON département = numéro
 WHERE population = 0
```

17) Dans l'algèbre relationnelle, la requête s'écrit

$$\gamma_{\texttt{modèle},\text{card}(*)\to\texttt{nb}}(\texttt{Assemblages})$$

et en SQL, elle se traduit ainsi.

```
SELECT modèle, COUNT(*) AS nb
 FROM     Assemblages
 GROUP BY modèle
```

18) Voilà pour le coût total (on multiplie le prix de chaque pièce par sa quantité).

```
SELECT modèle, SUM(coût * quantité) AS ctot
 FROM     Pièces JOIN Assemblages
           ON id = pièce
 GROUP BY modèle
```

Puis pour l'encombrement (même idée).

```
SELECT modèle, SUM(encombrement * quantité) AS etot
 FROM     Pièces JOIN Assemblages
           ON id = pièce
 GROUP BY modèle
```

Et enfin pour le rapport coût/encombrement.

```
SELECT modèle, SUM(coût * quantité) / SUM(encombrement * quantité) AS r
 FROM     Pièces JOIN Assemblages
           ON id = pièce
 GROUP BY modèle
```

20) Voici :

$$\gamma_{\texttt{Départements.nom}\to\texttt{nom},\text{somme}(\texttt{population})\to\texttt{pop}}\left(\sigma_{\texttt{Départements.nom}=\text{Haute-Saône}}\left(\texttt{Villes}\underset{\texttt{département}=\texttt{numéro}}{\bowtie}\texttt{Départements}\right)\right).$$

Dans la traduction en SQL, on n'est finalement pas obligé d'effectuer le regroupement, puisqu'après sélection il n'y a plus qu'un seul paquet, et qu'une fonction d'agrégation utilisée sans regroupement s'applique à la totalité des lignes.

```
SELECT Départements.nom AS nom, SUM(population) AS pop
 FROM     Villes JOIN Départements
           ON département = numéro
 WHERE    Départements.nom = "Haute-Saône"
 GROUP BY département
```

22) On fait une sélection en amont pour ne garder que les villes de plus de cent mille habitants, puis on fait une agrégation pour compter combien il y en a par département. Enfin, une sélection en aval permet de ne conserver que les départements où il y a au moins deux telles villes.

```
SELECT Départements.nom, COUNT(Villes.nom) AS nb
 FROM     Villes JOIN Départements
           ON département = numéro
 WHERE    population > 100000
 GROUP BY numéro
 HAVING   nb > 1
```

24) On commence par une requête qui donne la population pour chaque département. On la nommera `SR_1`.

```
SELECT département, SUM(population) AS p
 FROM villes
 GROUP BY département
```

Puis une requête qui donne la population de la préfecture de chaque département. On la nommera `SR_2`.

```
SELECT département.nom AS n, numéro, population
 FROM départements JOIN villes
       ON préfecture = id
```

On insère ces deux sous-requêtes dans une troisième, qui compare les populations.

```
SELECT n
 FROM (SR_1) AS t1 JOIN (SR_2) AS t2
       ON département = numéro
 WHERE population >= 0.1 * p
```

25) On utilise une requête qui donne la population de la ville la plus peuplée de chaque département.

```
SELECT département, MAX(population) AS m
 FROM villes
 GROUP BY département
```

Nommons cette requête `SR_1` et insérons-là dans une requête qui donne le nom de la ville correspondante. Et on ordonne dans le sens croissant.

```
SELECT villes.département, nom, population
 FROM villes JOIN (SR_1) AS t
       ON villes.département = t.département
 WHERE p = population
 ORDER BY population
```

Ça, c'est pour obtenir la ville la moins peuplée parmi les villes les plus peuplées de chaque département. Pour le contraire, on remplace le `MAX` par un `MIN`, et on ordonne dans le sens décroissant, mais à quoi bon ? On obtient évidemment Paris...

26) On commence par une requête qui donne la population de chaque département.

```
SELECT département, SUM(population) AS p
 FROM villes
 GROUP BY département
```

On la nomme `SR_1` et on l'insère dans une requête qui donne tous les couples de départements (distincts) et les sommes de populations correspondantes. Pour avoir les départements distincts, puisqu'on travaille avec leur numéro, il n'y a qu'à utiliser une comparaison stricte (ce qui évite d'avoir tous les couples en double). Deux remarques : on fait un produit cartésien (donc une jointure sans condition), et dans ce produit cartésien c'est deux fois la même table qui apparaît.

```
SELECT (t1.p + t2.p) AS sp
 FROM (SR_1) AS t1 JOIN (SR_1) AS t2
 WHERE t1.département < t2.département AND sp = 3000000
```

Si la requête produit au moins un résultat, la réponse à la question est « oui ». Si on veut en plus le nom des deux départements, il faut ajouter `t1.département` et `t2.département` en face du `SELECT`.

Pour trois département, on fait pareil, mais avec deux jointures. Ici comme il n'y a pas de condition, c'est facile.

```
SELECT (t1.p + t2.p + t3.p) AS sp
 FROM (SR_1) AS t1 JOIN (SR_1) AS t2 JOIN (SR_1) AS t3
 WHERE t1.département < t2.département
       AND t2.département < t3.département
       AND sp = 3000000
```

27) Commençons par une requête qui donne la quantité de chaque pièce lorsqu'on réunit deux modèles.

```
SELECT t1.modèle AS m1, t2.modèle AS m2,
       t1.pièce AS p, (t1.quantité + t2.quantité) AS q
 FROM Assemblages AS t1 JOIN Assemblages AS t2
       ON t1.pièce = t2.pièce
```

Insérons cette sous-requête, qu'on désignera `SR_1`, dans une requête qui, en regroupant par couples de modèles (m_1, m_2) (voilà un exemple de situation où le `GROUP BY` est suivi de deux attributs), calcule le minimum `mq` (sur l'ensemble des pièces utilisées) de la différence entre la quantité nécessaire dans 6450 et la quantité q disponible en regroupant m_1 et m_2. On ne conserve, dans cette agrégation, que les lignes pour lesquelles le minimum est positif (ce qui signifie qu'en assemblant 6450 à partir de m_1 et m_2, on ne manquera d'aucun type de pièce).

Au passage, on peut en profiter pour écarter les cas où $m_1 = 6450$ ou $m_2 = 6450$, qui ne sont pas très intéressants.

```
SELECT m1, m2, MIN(quantité - q) AS mq
 FROM Assemblages JOIN (SR_1) AS t
       ON pièce = p
 WHERE    modèle = 6301 AND m1 <> 6450 AND m2 <> 6450
 GROUP BY m1, m2
 HAVING   mq >= 0
```

28) On fait une requête qui calcule le nombre de naissances par jour.

```
SELECT jour, mois, année, COUNT(*) AS nb
 FROM naissances
 GROUP BY jour, mois, année
```

Puis une deuxième qui compte les naissances des filles uniquement.

```
SELECT jour, mois, année, COUNT(*) AS g
 FROM naissances
 WHERE genre = "F"
 GROUP BY jour, mois, année
```

Nommons respectivement ces deux requêtes `SR_1` et `SR_2` et insérons-les dans une requête qui teste l'égalité.

```
SELECT jour, mois, année
 FROM (SR_1) AS t1 JOIN (SR_2) AS t2
       ON t1.jour = t2.jour, t1.mois = t2.mois, t1.année = t2.année
 WHERE nb = g
```

Et on obtient tous les jours où ne sont nées que des filles.

31) On peut emprunter un titre si au moins l'un de ses exemplaires totalise le même nombre de sorties et d'entrées (autrement dit, à chaque fois qu'il a été emprunté, il a été rendu). Ou encore plus malin, on regarde si le nombre d'opérations concernant l'exemplaire est pair (un livre ne pouvant pas sortir deux fois de suite sans être entré dans l'intervalle).

```
SELECT MOD(COUNT(*), 2) AS m
 FROM     Ouvrages JOIN Opérations
           ON id = livre
 WHERE    titre = "L'homme qui rit" AND auteur = "Victor Hugo"
 GROUP BY livre
 HAVING   m = 0
```

L'emprunt est possible si et seulement si la requête ci-dessus renvoie une table non vide.

40) Dresser la table des prénoms et années correspondant à plus de mille naissances. En triant par ordre croissant de l'année, on obtiendra en tête de table le (ou les) prénom(s) ayant en premier établi le record.

```
SELECT prénom, année, COUNT(*) AS n
 FROM     Naissances
```

```
 GROUP BY prénom, année
 HAVING   n >= 1000
 ORDER BY année
```

Pour le jour, c'est plus compliqué parce qu'on ne dispose pas de moyen simple de ranger les dates (sous la forme utilisée dans ce tableau) dans l'ordre croissant. On peut utiliser douze requêtes (une par mois) et combiner les résultats (long), utiliser une fonction de manipulation des dates (pas au programme), ou bien trier seulement par année et chercher à vue la date la plus petite (peu satisfaisant et risqué : il se peut qu'il y ait beaucoup de lignes à regarder).

```
SELECT prénom, année, mois, jour, COUNT(*) AS n
 FROM     Naissances
 GROUP BY prénom, année, mois, jour
 HAVING   n >= 10
 ORDER BY année
```