Bases de données

Table des matières

I Introduction **2**
1 Nécessité de structures de données . . . 2
2 Principe des bases de données . . . 2

II Modèle relationnel **3**
1 Définitions . . . 3
2 Opérateurs de l'algèbre relationnelle . . . 6
a Notation ensembliste . . . 6
b SQL . . . 6
c Projection . . . 7
d Sélection . . . 9
e Jointures . . . 10
f Renommage . . . 14
g Opérations ensemblistes . . . 14
h Produit cartésien . . . 16
i Division cartésienne . . . 18
j Fonctions d'agrégat . . . 19

III Compléments de SQL **20**
1 Ordre, limite, offset . . . 20
2 Requêtes avec blocs imbriqués . . . 20
a Si la requête interne ne renvoie qu'un résultat . . . 21
b Si la requête interne renvoie plusieurs résultats . . . 21
c Existence . . . 22
3 Regroupement avant agrégation : GROUP BY et HAVING . 24
a GROUP BY . . . 24
b HAVING . . . 25
c Agrégats de double niveau . . . 25
4 Cas de la division . . . 26

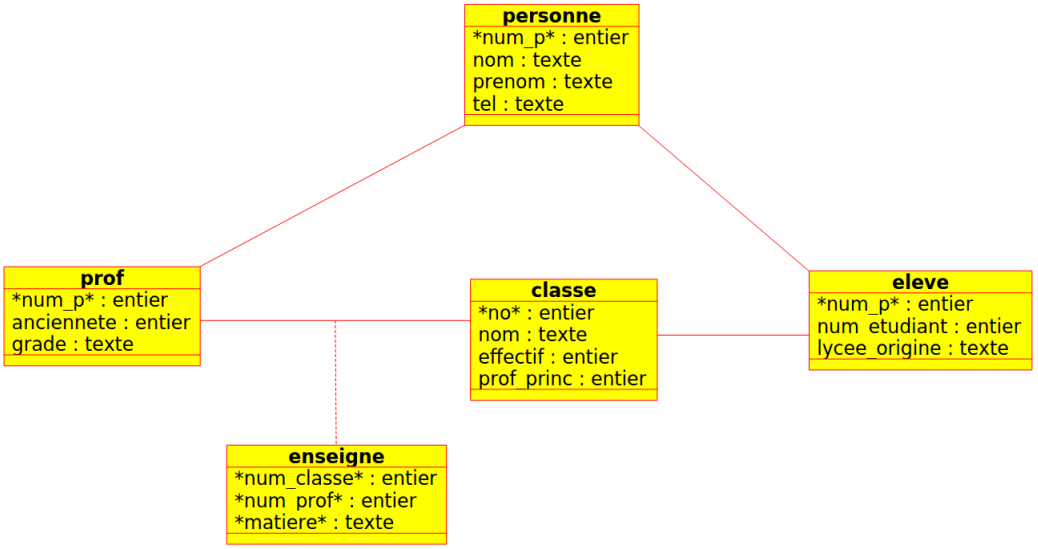
I INTRODUCTION

1 Nécessité de structures de données

Le lycée souhaite créer un fichier de renseignement sur les élèves. Sur chaque fiche, on inscrit des renseignements : nom, prénom, adresse, lycée d'origine, nom des professeurs, etc.

Problèmes :

- *homonymie* : si plusieurs élèves ont même nom, prénom ?
  Solution : ajouter un numéro d'élève.
- Comment trouver tous les élèves ayant un même enseignant ? d'une même classe ? Si on doit mettre à jour la classe ou le nom d'un professeur ? Faut-il tout refaire ?
  Solution : séparer les données.



(Il manque un num_classe dans la table eleve)

2 Principe des bases de données

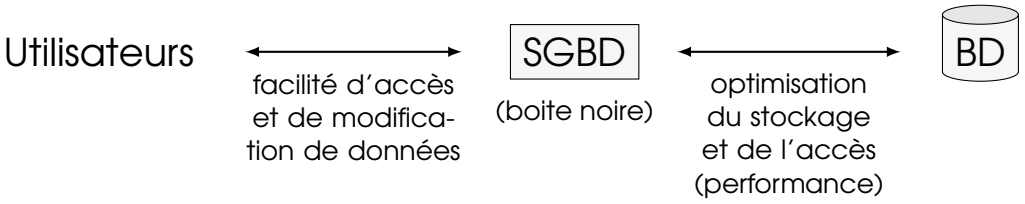
Une base de données est un ensemble d'informations qui doit être

- **cohérent** : élimination de redondances.
- **partagé** : utilisable par plusieurs utilisateurs.

Le cycle de vie d'une base de données se sépare en trois phases :

- **conception** : le plus difficile (Hors programme [1])
- **implémentation** : via un SGBD (système de gestion de base de données) : définition des tables et insertion des données. (Hors programme)
- **Utilisation** : extraction de données, mise à jour (insertion, modification, destruction).

Les deux derniers points se font à l'aide d'un langage appelé SQL (Structured Query Language).



II MODÈLE RELATIONNEL

1 Définitions

Les objets et leurs associations sont représentés par la notion de **relations** (on dit aussi **tables**) qui sont des tableaux à deux dimensions.

Exemple

clé primaire ↓

personne :	**num_p**	nom	prenom	tel	← Attributs
	(entier)	(texte)	(texte)	(texte)	← Domaines
	6275	Dupont	Pierre	0645456789	← Tuple
	1234	Durand	Sylvain	0612253694	

clé étrangère ↓

classe :	**no**	nom	effectif	prof_princ
	(entier)	(texte)	(entier)	(entier)
	12	MPSI 3	40	6275
	22	PCSI 2	30	5781

1. Il est écrit précisément dans les principales capacités développées dans cette partie de la formation : concevoir une base constituée de plusieurs tables, et utiliser les jointures symétriques pour effectuer des requêtes croisées. Aucune subtilité sur la cohérence et l'optimisation d'une base de donnée n'est au programme.

Définition

Étant donnée une relation (table), on appelle :

- **Attributs** : les colonnes qui la composent.
- **Domaine d'un attribut** : ensemble de valeurs que peut prendre un attribut (entier, réel, chaîne de caractères, etc.)
- **Tuple** : Ligne d'une relation.
- **Clé primaire** : Ensemble minimal d'attributs qui permet d'identifier un tuple unique de la relation. On parle aussi d'identifiant.
- **Schéma** : Nom de la relation suivi de la liste des attributs et de leur domaine. Clé primaire à souligner (en gras dans ce poly).
- **Clé étrangère** : Ensemble d'attributs égal à la clé primaire d'une autre relation : un tuple de la relation ne peut exister que si sa clé étrangère est égale à la clé primaire d'un tuple dans l'autre relation.

Remarque

Une relation est donc mathématiquement un ensemble de n-uplets appartenant à $D_1 \times \cdots \times D_n$ où $D_1, \ldots, D_n$ sont les domaines de ses attributs, c'est-à-dire une partie de $D_1 \times \cdots \times D_n$.

Exemple : Schéma de la base de donnée Lycée (gras pour les clés primaires, flèches pour les clés étrangères)

- personne
  - ⋆ **num_p** : entier
  - ⋆ nom : texte
  - ⋆ prenom : texte
  - ⋆ tel : texte
- eleve
  - ⋆ **num_p** : entier → personne.num_p
  - ⋆ num_etudiant : entier
  - ⋆ lycee_origine : texte
  - ⋆ num_classe : entier → classe.no
- classe
  - ⋆ **no** : entier
  - ⋆ nom : texte
  - ⋆ effectif : entier
  - ⋆ prof_princ : entier → prof.num_p
- prof
  - ⋆ **num_p** : entier → personne.num_p
  - ⋆ ancienete : entier
  - ⋆ grade : texte
- enseigne
  - ⋆ **num_classe** : entier → classe.no
  - ⋆ **num_prof** : entier → prof.num_p
  - ⋆ **matiere** : texte

Définition

La **population** d'une relation est l'ensemble des tuples de la relation.

Exemple : Populations

- classe :

no	nom	effectif	prof_princ
12	MPSI 3	40	6275
22	PCSI 2	30	5781
5	MP*	22	1234

- prof :

num_p	anciennete	grade
6275	20	Agr
5781	5	Agr HC
1234	10	Agr CS

- enseigne :

num_classe	num_prof	matiere
12	6275	Maths
12	5781	Info
22	5781	Chimie
5	1234	Maths

- personne :

num_p	nom	prenom	tel
6275	Dupont	Pierre	0645456789
1234	Durand	Sylvain	0612253694
5781	Martin	Pierre	0635147630
2533	Galois	Evariste	0699999999
2517	Martin	Pierre	NULL
1254	Beauval	Elise	0678951354
5544	Martin	Filou	0612345678

- eleve :

num_p	num_etudiant	lycee_origine	num_classe
5544	12	Henry-le-petit	12
2533	45	Louis VI	5
2517	58	Carnot	22
1254	22	Carnot	12

Tous les attributs sont **mono-valués** : ils ne peuvent prendre qu'une seule valeur (exemple des professeurs d'une classe : on a créé une table `enseigne` plutôt que des les faire apparaître dans `classe`, exemple de plusieurs prénoms...)

Lorsque l'on ne peut pas préciser un attribut (exemple : nom de jeune fille), on lui attribue la valeur NULL.

⚠ un identifiant (clé primaire) ne peut pas être NULL !

Cela peut poser des problèmes.

Exemple

Soit une table Employé(**n°**, nom, salaire, commission) et deux tuples (7, DURAND, 1500, 200) et (6, DESCHAMPS, 1300, NULL).

A la fin du mois, on obtient pour le premier $1500 + 200 = 1700$ et pour le second 1300+NULL=NULL...

Difficulté : Construire le schéma d'une relation de manière optimale.

Bonne nouvelle : c'est hors programme.

2 Opérateurs de l'algèbre relationnelle

a Notation ensembliste

Notation

Une relation d'attributs $A_1,\dots,A_n$ est notée $R(A_1,\dots,A_n)$ où R est le nom de la relation.

Si t désigne un tuple de cette relation, on note $t \in R$.

La valeur de l'attribut A_i dans t se note $t.A_i$ ou $t[A_i]$.

Le p-uplets des attributs $A_{i_1},\dots,A_{i_p}$ dans t se note $t\left[A_{i_1},\dots,A_{i_p}\right]$.

Exemple

Dans la relation personne(num_p, nom, prenom, tel), $t = (123,'Titi','Toto','0123456789')$.

$t \in personne$.

$t[nom] =' Titi'$

$t[num\_p, prenom] = (123,'Toto')$.

Remarques

R1 – Si X désigne une clé primaire (un ou plusieurs attributs), alors

$$\forall\, t,t' \in R \mid t \neq t',\quad t[X] \neq t'[X] \qquad et \qquad \forall\, t \in R,\quad t[X] \neq \text{NULL}.$$

R2 – Soit $R(A_1,\dots,A_n)$. Pour tous $t,t' \in R$,

$$t \neq t' \iff \exists\, i,\quad t[A_i] \neq t'[A_i].$$

b SQL

L'algèbre relationnelle permet de formaliser les requêtes alors que le SQL est le langage permettant d'interroger effectivement la base de donnée.

Les requêtes se présentent sous l'une des formes :

```
SELECT ...
FROM ...
WHERE ...
;
```

```
SELECT ...
FROM ... JOIN ... ON ...
WHERE ...
;
```

Par convention on notera en majuscules les instructions (mais le SQL n'est pas sensible à la casse.) Les instructions SQL se terminent toujours par un point-virgule.

L'instruction de base

```
SELECT * FROM eleve;
```

permet de renvoyer tous les tuples de la table `eleve`.

C Projection

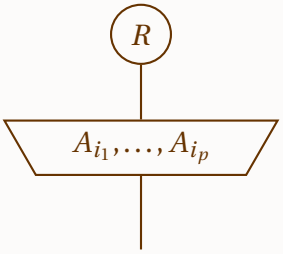
Définition : Projection

Soit $R(A_1,\dots A_n)$ une relation d'attributs $A_1,\dots,A_n$.
On appelle **projection** de R sur $(A_{i_1},\dots,A_{i_p})$ la relation notée $\Pi_{A_{i_1},\dots,A_{i_p}}(R)\,(A_{i_1},\dots,A_{i_p})$ obtenue en ne gardant que les attributs $A_{i_1},\dots,A_{i_p}$ dans tous les tuples de R.

$$\Pi_{A_{i_1},\dots,A_{i_p}}(R) = \left\{t\left[A_{i_1},\dots,A_{i_p}\right]\ ;\ t\in R\right\}.$$

Remarque

La projection est symbolisée par :



Les diagrammes symbolisant les requêtes en algèbre relationnelle ne sont pas au programme mais sont utiles pour visualiser simplement ces requêtes.

Exemple

Dans la base lycée, $\Pi_{nom,prenom}(personne)$ est la relation :

nom	prenom
Dupont	Pierre
Durand	Sylvain
Martin	Pierre
Galois	Evariste
Martin	Pierre
Beauval	Elise
Martin	Filou

Remarque

⚠ Mathématiquement, on manipule des ensembles, donc il n'y a pas de répétition.

En SQL :

```
SELECT A1,...,Ap FROM R;
```

⚠ Pas d'ensemble en SQL (trop coûteux) : il peut y avoir des répétitions. Pour les éviter :

```
SELECT DISTINCT A1,...,Ap FROM R;
```

Pour l'exemple précédent :

```
SELECT nom, prenom FROM personne;
```

On peut aussi renommer les attributs et les tables. Par exemple,

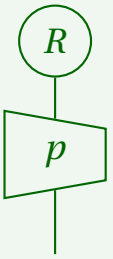
```
SELECT p.nom AS "Nom de la personne", p.prenom AS "Prénom"
FROM personne p;
```

d Sélection

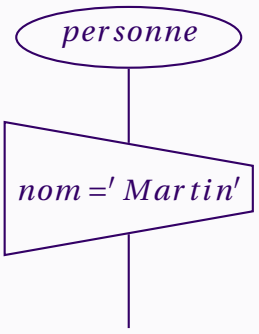
Définition : Selection

Soit $R(A_1,\dots,A_n)$ une relation. La sélection de R par le prédicat p notée $\sigma_p(R)(A_1,\dots,A_n)$ est la relation constituée des tuples de R vérifiant p.

$$\sigma_p(R) = \{t \in R \mid t \text{ satisfait } p\}$$



Exemple : Quelles sont les personnes nommées Martin ?



$\sigma_{nom='Martin'}(personne)$

num_p	nom	prenom	tel
5781	Martin	Pierre	0635147630
2517	Martin	Pierre	NULL
5544	Martin	Filou	0612345678

Toutes les opérations logiques sont possibles : =, ≠, <, >, ⩽, ⩾, ∧ (et), ∨ (ou), ¬ (non).

En SQL :

```
SELECT * FROM R WHERE p;
```

Liste non exhaustive de comparateurs :

=	!= ou <>	<	>
<=	>=	IS NULL	IS NOT NULL
AND	OR	NOT	BETWEEN ... AND ...

Citons aussi `LIKE '...'` (_ remplace un caractère, % remplace un nombre quelconque de caractères.)

Exemples

E1 – `LIKE 'N%'` : commence par N.

E2 – `LIKE '%a%'` : contient a.

E3 – `LIKE 'pa%on'` : commence par pa, finit par on (paon, pantalon, passion, etc.)

E4 – numéros des personnes qui s'appellent Martin avec un prénom est différent de Pierre ou dont le nom commence par D :

```
SELECT num_p FROM personne
WHERE (nom = 'Martin' AND prenom != 'Pierre')
      OR (nom LIKE 'D%');
```

e Jointures

Définition : Jointure

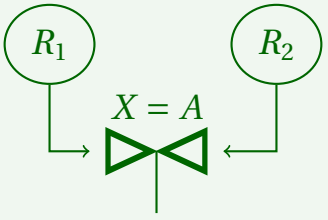
Soient $R_1(X,Y)$ et $R_2(A,B)$ deux relations avec X, Y, A, B attributs ou ensembles d'attributs tels que $X \neq \varnothing$ et $A \neq \varnothing$.

La **jointure** de R_1 et R_2 selon la condition $X = A$ (ou $R_1.X = R_2.A$) est la relation

$$R_1 \underset{R_1.X=R_2.A}{\bowtie} R_2(X,Y,A,B)$$

dont les tuples sont de la forme (x,y,a,b) tels que $x = a$, $(x,y) \in R_1$ et $(a,b) \in R_2$.

$$R_1 \underset{X=A}{\bowtie} R_2 = \{t \mid t[X,Y] \in R_1 \ ; \ t[A,B] \in R_2 \ ; \ t[X] = t[A]\}$$



Remarque

Seules ces jointures (appelée **équijointures**) sont au programme, mais il en existe bien d'autres. On peut par exemple mettre autre chose qu'une égalité dans la condition.

En SQL :

```sql
SELECT *
FROM R1 JOIN R2
        ON X = A;
```

```sql
SELECT *
FROM R1, R2
WHERE X = A;
```

En théorie, la seconde est moins optimisé (voir paragraphe produit cartésien).

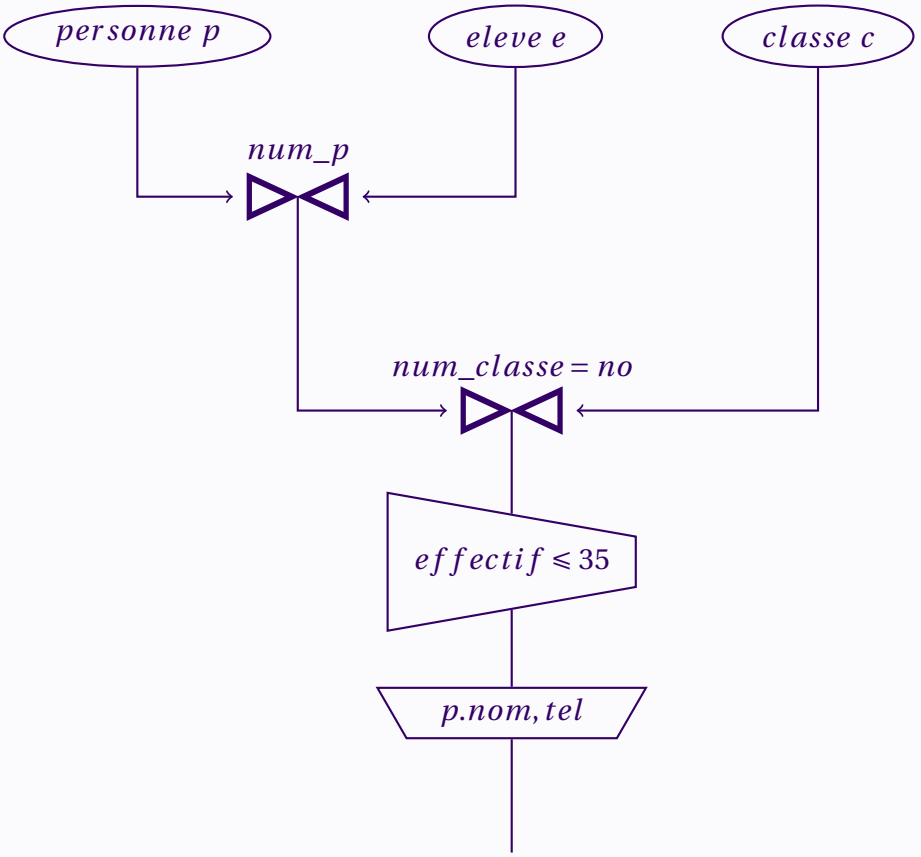
Exemples

E1 – Jointure de personne et prof sur num_p : $personne \underset{num\_p}{\bowtie} prof$.

personne.num_p	nom	prenom	tel	prof.num_p	anciennete	grade
6275	Dupont	Pierre	0645456789	6275	20	Agr
1234	Durand	Sylvain	0612253694	1234	10	Agr CS
5781	Martin	Pierre	0635147630	5781	5	Agr HC

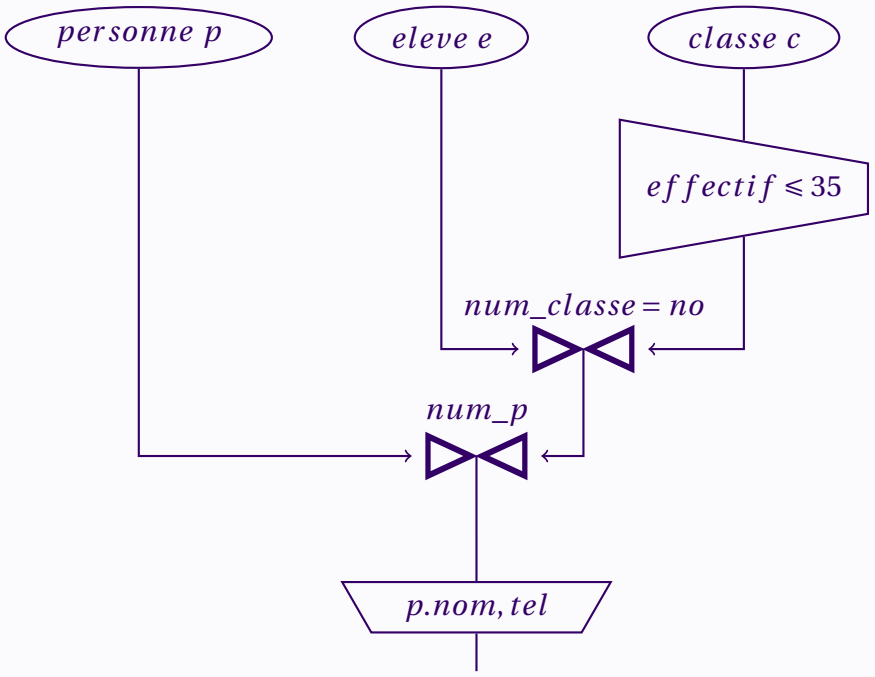
```sql
SELECT *
FROM personne pe JOIN prof pr ON pe.num_p = pr.num_p;
```

E2 – Donner le nom et le numéro de téléphone des étudiants dont l'effectif de la classe est inférieur ou égal à 35.



$$\Pi_{p.nom,tel}\ \sigma_{effectif \leqslant 35}\Big((classe\ c) \underset{num\_classe=no}{\bowtie} \Big((eleve\ e) \underset{num\_p}{\bowtie} (personne\ p)\Big)\Big)$$

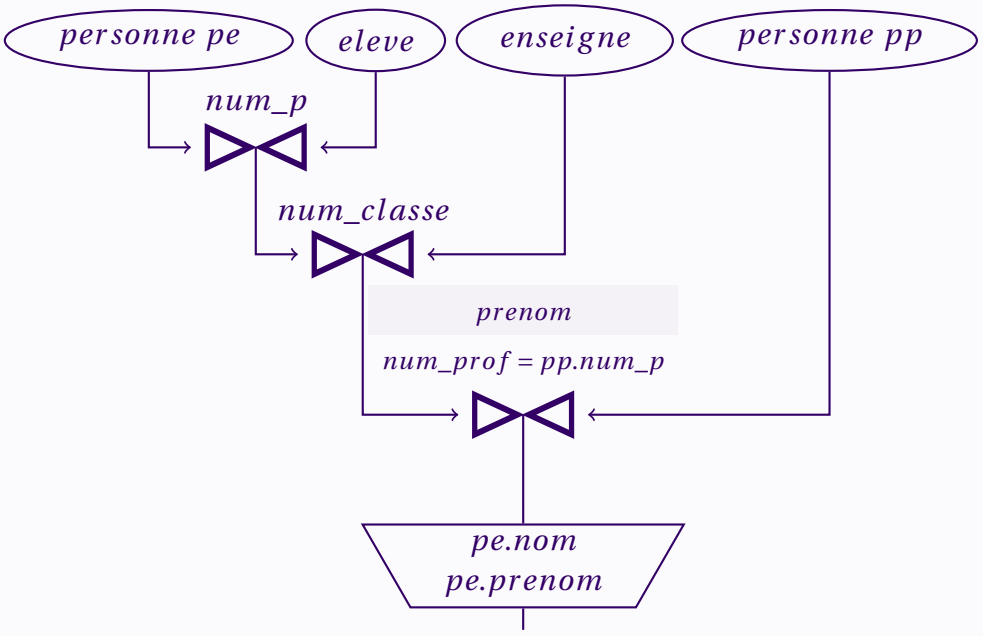
Version plus optimisée ? On peut faire la sélection avant la jointure.



$$\Pi_{p.nom,tel}\Big((personne\ p) \underset{num\_p}{\bowtie} \Big((eleve\ e) \underset{num\_classe=no}{\bowtie} \sigma_{effectif \leqslant 35}(classe\ c)\Big)\Big)$$

```
SELECT p.nom, tel
FROM classe c JOIN eleve e      ON num_classe = no
              JOIN personne p   ON p.num_p = e.num_p
WHERE effectif <= 35;
```

E3 – Nom et prénom des élèves ayant le même prénom qu'un de leurs professeurs ?

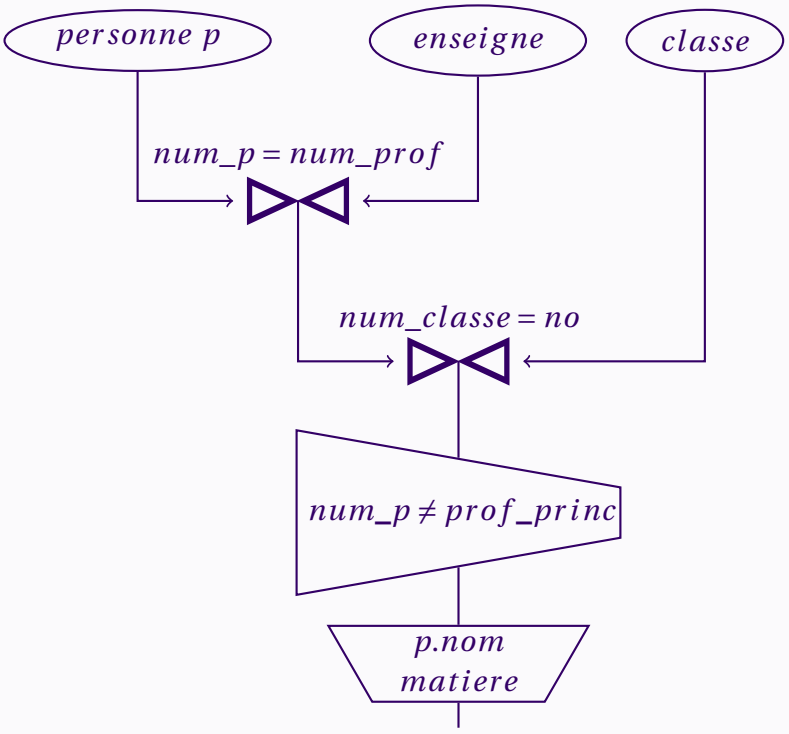


$$T = \left(\left(personne\ pe \underset{num\_p}{\bowtie} eleve\right) \underset{num\_classe}{\bowtie} enseigne\right) \underset{\substack{prenom \\ num\_prof = pp.num\_p}}{\bowtie} personne\ pp$$

$$\Pi_{pe.nom,pe.prenom}(T)$$

```
SELECT pe.nom, pe.prenom
FROM personne pe JOIN eleve e     ON e.num_p = pe.num_p
                 JOIN enseigne en ON en.num_classe
                                     = el.num_classe
                 JOIN personne pp ON pp.prenom = pe.prenom
                                 AND pp.num_p = num_prof;
```

E4 – Nom et matière des professeurs enseignant dans une classe dont ils ne sont pas professeurs principaux ?



$$T = classe \underset{num\_classe=no}{\bowtie} \left(enseigne \underset{num\_prof=num\_p}{\bowtie} personne\ p\right)$$

$$\Pi_{p.nom,matiere} \cdot \sigma_{num\_p \neq prof\_princ}(T)$$

```
SELECT p.nom, matiere
FROM personne p JOIN prof pr  ON pr.num_p = p.num_p
                JOIN enseigne ON num_prof = p.num_p
                JOIN classe   ON no = num_classe
WHERE p.num_p != prof_princ;
```

f Renommage

Définition : Renommage

Soit $R(X, Y)$ une relation, X, Y attributs ou ensembles d'attributs.
Si A et X ont même taille et si A n'apparaît pas dans le schéma de R, $\rho_{X\to A}(R)(A, Y)$ est la relation de schéma (A, Y) telle que

$$\rho_{X\to A}(R) = \{t \ ; \ \exists\, t' \in R, \quad t[A] = t'[X] \text{ et } t[Y] = t'[Y]\}$$

obtenue en renommant X en A.

Remarque

Utile pour les opérations ensemblistes qui demandent d'avoir les mêmes schémas, par exemple.

En SQL :

```
SELECT nom_attribut AS nouveau_nom_attribut
FROM nom_table nouveau_nom_table
WHERE ...
;
```

ou

```
SELECT nom_attribut AS nouveau_nom_attribut
FROM nom_table AS nouveau_nom_table
WHERE ...
;
```

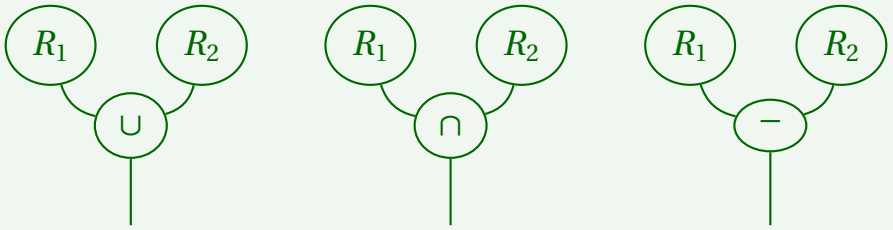
g Opérations ensemblistes

Définition : Opérations ensemblistes

Soient deux relations **de même schéma** $R_1(A_1, \ldots, A_n)$ et $R_2(A_1, \ldots, A_n)$.
On définit les relations de schéma $(A_1, \ldots, A_n)$ également :

- $R_1 \cup R_2 = \{t \mid t \in R_1 \text{ ou } t \in R_2\}$
- $R_1 \cap R_2 = \{t \mid t \in R_1 \text{ et } t \in R_2\}$
- $R_1 - R_2 = \{t \mid t \in R_1 \text{ et } t \notin R_2\}$

Symbolisées par :



Exemple

$\mathbf{R_1}$:	**nom**	**prenom**
	Martin	Jean
	François	Marie

et

$\mathbf{R_2}$:	**nom**	**prenom**
	Martin	Jean
	François	François
	Marie	Luc

Alors

$\mathbf{R_1 \cup R_2}$:	**nom**	**prenom**
	Martin	Jean
	François	Marie
	François	François
	Marie	Luc

$\mathbf{R_1 \cap R_2}$:	**nom**	**prenom**
	Martin	Jean

$\mathbf{R_1 - R_2}$:	**nom**	**prenom**
	François	Marie

Remarque

L'intersection s'obtient à l'aide de la soustraction : $R_1 \cap R_2 = R_1 - (R_1 - R_2) = R_2 - (R_2 - R_1)$.

En SQL :

```
SELECT ... { INTERSECT | EXCEPT | UNION | UNION ALL } SELECT ... ;
```

⚠ Pas de parenthèses autour des `SELECT` ! Dans certains SGBD (MySQL,

Oracle, ...), on rencontre `MINUS` à la place d'`EXCEPT`.

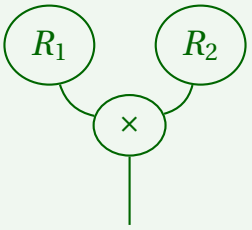
`UNION ALL` permet de garder les répétitions lors du calcul de la réunion.

h Produit cartésien

Définition : Produit cartésien

Soient $R_1(A_1,\dots,A_n)$ et $R_2(B_1,\dots,B_p)$ n'ayant aucun attribut en commun. $R_1 \times R_2(A_1,\dots,A_n,B_1,\dots,B_p)$ est la relation obtenue en concaténant tout tuple de R_1 avec tout tuple de R_2.

$$R_1 \times R_2 = \{t \ ; \ t[A_1,\dots,A_n] \in R_1 \ \text{ et } \ t[B_1,\dots,B_p] \in R_2\}$$



Remarque

Si R_1 a M tuples et R_2 a N tuples, alors $R_1 \times R_2$ en a $M \times N$.

Exemples

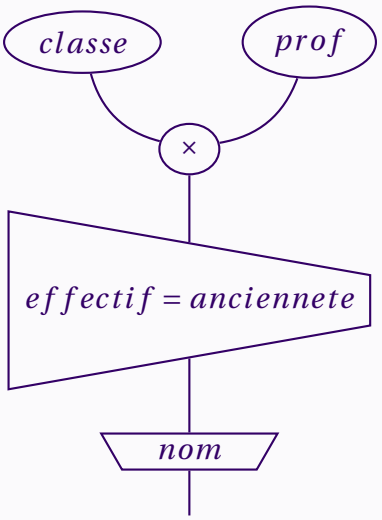
E1 –

R_1 : A	B
0	1
3	4

R_2 : C	D
5	6
7	8

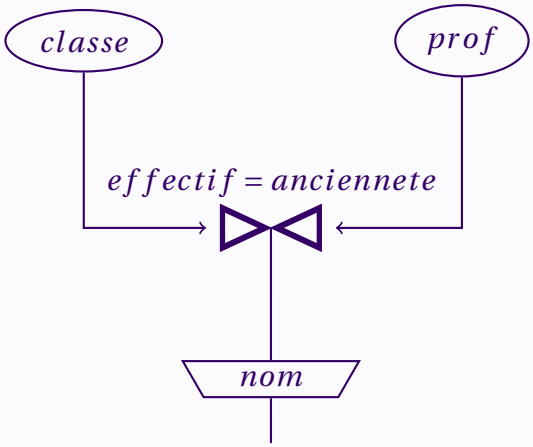
$R_1 \times R_2$: A	B	C	D
0	1	5	6
3	4	5	6
0	1	7	8
3	4	7	8

E2 – Quel est le nom des classes dont l'effectif est égal à l'ancienneté d'un enseignant ?



$$\Pi_{nom}\,\sigma_{effectif=anciennete}(classe \times prof)$$

peut aussi se faire avec une jointure (préférable) :



$$\Pi_{nom}\left(classe \underset{effectif=anciennete}{\bowtie} prof\right)$$

En SQL :

```
SELECT * FROM R1, R2;
```

Remarques

R1 – Lorsque l'on fait une jointure avec :

```
SELECT ... FROM R1, R2 WHERE R1.X=R2.A;
```

on fait a priori une sélection sur le produit cartésien qui coûte cher à calculer. On se doute que l'optimiseur du SGBD va faire en sorte de ne pas calculer complètement le produit cartésien, mais la formulation :

```
SELECT ... FROM R1 JOIN R2 ON R1.X=R2.A;
```

est souvent préférée pour lever toute ambiguïté et faire apparaître clairement les jointures.

R2 – Il n'est pas necessaire en SQL que les attributs de R_1 et R_2 soient tous différents (ceux de R_1 sont préfixés par « $R_1.$ » et ceux de R_2 par « $R_2.$ ».)

Division cartésienne

Définition

Soient $R_1(A_1,\dots,A_n)$ et $R_2(A_1,\dots,A_p)$ où $p<n$.
On définit $R_1 \div R_2(A_{p+1},\dots,A_n)$ telle que $R_1 \div R_2 = \{t\ ;\ \forall\ t' \in R_2,\ \ (t,t') \in R_1\}$.

Remarque

Souvent utile pour traduire des requêtes avec « tous ». (Mais attentions, certains « tous » se traduisent par une différence ensembliste.

Exemples

E1 –

R_1 :	A	B	C
	1	1	1
	1	2	0
	1	2	1
	1	3	0
	2	1	1
	2	3	3
	3	1	1
	3	2	0
	3	2	1

R_2 :	B	C
	1	1
	2	0

$R_1 \div R_2$:	A
	1
	3

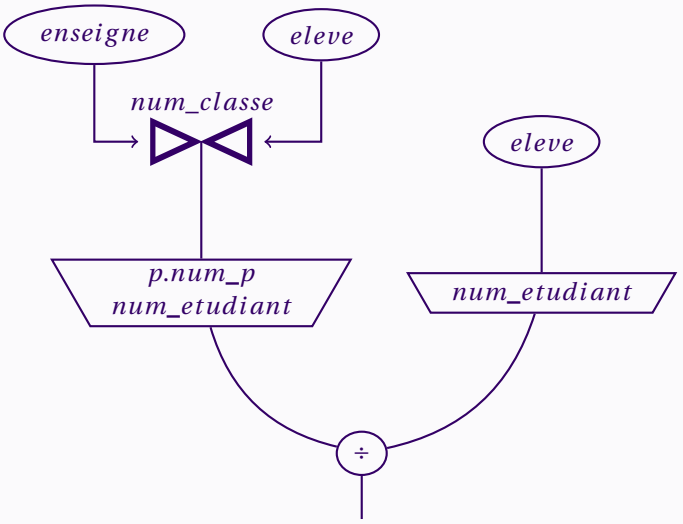
R'_2 :	B	C
	1	1

$R_1 \div R'_2$:	A
	1
	2
	3

R''_2 :	B	C
	3	5

$R_1 \div R''_2$:	A
	/

E2 – Numéro des profs enseignants à tous les élèves ?



$$\left(\Pi_{p.num\_p,num\_etudiant}\left(eleve \underset{num\_classe}{\bowtie} enseigne\right)\right) \div \left(\Pi_{num\_etudiant}(eleve)\right)$$

Remarque

Trop coûteuse, la division n'est pas implémentée en SQL. Nous verrons plus loin comme la contourner.

j Fonctions d'agrégat

Il est possible d'appliquer des fonctions aux attributs d'une table au sens où on applique la fonction à cet attribut dans chaque tuple.

Exemple : conversion monétaire dans une table article(num, nom, prix_unitaire, nb)

```
SELECT nom, prix_unitaire * 1.11 AS prix_unitaire_usd FROM article;
```

Exemple : Prix total des articles :

```
SELECT nom, prix_unitaire * nb AS prix_total FROM article;
```

Il est également possible d'appliquer des fonctions utilisant toutes valeurs des tuples d'une table, appelées **fonctions d'agrégat** : somme, minimum, maximum, moyenne, décompte (et bien d'autres hors programme).

En SQL : `SUM, MIN, MAX, AVG, COUNT.`

Exemple : Moyenne des anciennetés des professeurs

```
SELECT AVG(anciennete) FROM prof;
```

Exemple : Nombre d'élèves dont le nom est Martin

```
SELECT COUNT(*)
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE p.nom = 'Martin';
```

Remarque

Pour supprimer les doublons : `COUNT (DISTINCT ...).`

III COMPLÉMENTS DE SQL

1 Ordre, limite, offset

La clause `ORDER BY A` permet de trier les résultats par A croissants ; `ORDER BY A DESC` par A décroissants.

Exemple :

```
SELECT * FROM personne ORDER BY nom, prenom;
SELECT * FROM prof ORDER BY anciennete DESC;
```

La clause `LIMIT n` permet de limiter aux n premiers tuples.

La clause `LIMIT n OFFSET m` permet de renvoyer les n premiers tuples suivant les m premiers tuples (du numéro $m+1$ au numéro $m+n$.)

Exemple : Deuxième plus ancien professeur, en supposant que chaque ancienneté n'est obtenue que par un professeur

```
SELECT * FROM prof ORDER BY anciennete DESC LIMIT 1 OFFSET 1;
```

2 Requêtes avec blocs imbriqués

Remarque

Sans renommage et en cas d'homonymie, les attributs sont ceux de la table la plus interne.

a Si la requête interne ne renvoie qu'un résultat

$$\texttt{SELECT ... FROM ... WHERE expr} \left\{ \begin{array}{c} = \\ != \\ < \\ > \\ <= \\ >= \end{array} \right\} \underbrace{\texttt{(SELECT ...)}}_{\text{ne doit renvoyer qu'un résultat.}} \texttt{;}$$

Exemple : Professeur(s) le(s) plus jeune(s)

```
SELECT num_p
FROM prof
WHERE anciennete = (SELECT MIN(anciennete) FROM prof);
```

b Si la requête interne renvoie plusieurs résultats

On utilise alors :

- `IN ...` ($\in$)
- `NOT IN ...` ($\notin$)
- $\left\{ \begin{array}{c} = \\ != \\ < \\ > \\ <= \\ >= \end{array} \right\}$ `ANY ...` ($\exists$)
- $\left\{ \begin{array}{c} = \\ != \\ < \\ > \\ <= \\ >= \end{array} \right\}$ `ALL ...` ($\forall$).

Remarques

R1 – `ALL` et `ANY` ne fonctionnement pas en `SQLite`.

R2 – `IN` $\Longleftrightarrow$ `= ANY`.

R3 – `NOT IN` $\Longleftrightarrow$ `!= ALL`.

R4 – `!=ANY` $\Longleftrightarrow$ `NOT (... = ALL)` : ne s'utilise jamais !

Exemples

E1 – Numéros des professeurs de MPSI

```
SELECT num_prof
FROM enseigne
WHERE num_classe IN (SELECT no FROM classe
                     WHERE nom LIKE 'MPSI%');
```

(préférer une jointure dans ce cas.)

E2 – numéro des fournisseurs qui livrent le même produit que le fournisseur n° 1 en plus grande quantité (voir schéma dans le TD).

```
SELECT nF
FROM L L1
WHERE nP IN (SELECT nP FROM L
             WHERE nF = 1 AND L1.quantite >= quantite);
```

E3 – Ville des commerces n° 1, 2, 3.

```
SELECT ville FROM C WHERE nC IN (1,2,3);
```

E4 – Professeurs les plus jeunes

```
FROM prof
WHERE anciennete <= ALL (SELECT anciennete FROM prof);
```

C Existence

```
SELECT ... FROM ... WHERE EXISTS (SELECT ...);
SELECT ... FROM ... WHERE NOT EXISTS (SELECT ...);
```

Exemple : Nom des élèves n'étant pas seuls issus d'un lycée

```
SELECT nom
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE EXISTS (SELECT * FROM eleve
              WHERE lycee_origine = e.lycee_origine
                AND num_p != e.num_p);
```

ou bien

```
SELECT nom
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE 1 < (SELECT COUNT(*) FROM eleve
           WHERE lycee_origine = e.lycee_origine);
```

ou encore

```
SELECT nom
FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE lycee_origine = ANY (SELECT lycee_origine FROM eleve
            WHERE num_p != e.num_p);
```

ou encore avec une jointure (à préférer)

```
SELECT nom
FROM personne p JOIN eleve e1 ON e1.num_p = p.num_p
                JOIN eleve e2 ON e2.lycee_origine=e1.lycee_origine
WHERE e1.num_p != e2.num_p;
```

Remarque

Un `EXISTS (...)` équivaut à un `0 != (SELECT COUNT(*) ...)`.

Exemple : Nom des professeurs non professeurs principaux

```
SELECT nom
FROM personne pp JOIN prof p ON pp.num_p = p.num_p
WHERE NOT EXISTS (SELECT * FROM classe WHERE num_p = prof_princ);
```

ou bien

```
SELECT nom
FROM personne pp JOIN prof p ON pp.num_p = p.num_p
WHERE p.num_p NOT IN (SELECT prof_princ FROM classe);
```

ou encore

```
SELECT nom
FROM personne pp
JOIN prof p ON pp.num_p = p.num_p
EXCEPT
SELECT nom
FROM personne
JOIN classe ON prof_princ = num_p;
```

3 Regroupement avant agrégation : GROUP BY et HAVING

a GROUP BY

Idée : regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat.

```
SELECT ... FROM ... WHERE ... GROUP BY ...;
```

Résultat : Une seule ligne est renvoyée par groupe.

Cohérence : Il faut donc que le contenu du `SELECT` soit cohérent : soit des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe, soit des fonctions d'agrégat (qui porteront sur chaque groupe).

Exemples

E1 – Déterminer le nombre moyen d'élèves de chaque professeur.

```
SELECT num_prof, AVG(effectif) AS "Effectif moyen"
FROM classe JOIN enseigne ON num_classe = no
GROUP BY num_prof;
```

Mais attention, les éventuels professeurs n'ayant pas d'élèves (???) n'apparaîtront pas.

Pour les voir, on peut par exemple faire :

```
SELECT num_prof, 0 AS "Effectif moyen" FROM prof
WHERE num_prof NOT IN (SELECT num_prof from enseigne)

UNION

SELECT num_prof, AVG(effectif) AS "Effectif moyen"
FROM classe JOIN enseigne ON num_classe = no
GROUP BY num_prof;
```

E2 – Combien de produits différents ont été livrés par chaque fournisseur ?

```
SELECT nF, COUNT(DISTINCT nP)
FROM L
GROUP BY nF;
```

(À nouveau, sans les fournisseurs qui n'ont fait aucune livraison...)

b HAVING

Idée : On peut imposer une condition sur les groupes définis par `GROUP BY` :

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...;
```

Cohérence : Comme pour le `SELECT`, dans la condition du `HAVING` doivent apparaître soit des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe, soit des fonctions d'agrégat (qui porteront sur chaque groupe).

Exemples

E1 – Nom et nombre moyen d'élèves de chaque professeur qui enseigne dans au moins deux classes

```
SELECT p.nom, AVG(effectif)
FROM enseigne JOIN classe ON num_classe = no
              JOIN personne p ON num_prof = num_p
GROUP BY num_prof HAVING COUNT(DISTINCT num_classe) > 1;
```

E2 – Nombre de produits différents livré par chaque fournisseur tel que ce fournisseur livre une quantité totale plus grande que 1 000 unités

```
SELECT nF, COUNT(DISTINCT nP)
FROM L
GROUP BY nF HAVING SUM(quantite) > 1000;
```

c Agrégats de double niveau

Un `GROUP BY` renvoyant plusieurs résultats, on peut enchaîner avec une autre fonction d'agrégat.

Exemples

E1 – Moyenne du nombre total d'élèves par professeur (ayant des élèves)

```
SELECT AVG(somme)
FROM (SELECT num_prof, SUM(effectif) AS somme
      FROM classe JOIN enseigne ON num_classe = no
      GROUP BY num_prof);
```

Mais on a compté plusieurs fois une même classe si le professeur enseigne plusieurs matières dans la classe. Pour éviter cela, on peut écrire :

```sql
SELECT AVG(somme)
FROM
  ( SELECT num_prof, SUM(effectif) AS somme
    FROM
     ( SELECT num_prof, effectif
       FROM enseigne JOIN classe ON num_classe = no
       GROUP BY num_prof, num_classe )
    GROUP BY num_prof );
```

E2 – Nombre maximum de livraisons effectuées par un fournisseur.

```sql
SELECT MAX(nb)
FROM (SELECT COUNT(*) as nb
      FROM L
      GROUP BY nF);
```

4 Cas de la division

La division n'existe pas en SQL. Voici plusieurs manières de l'obtenir :

- Avec un `NOT EXISTS` et un `NOT IN` :

$$\begin{aligned} x \in R_1 \div R_2 &\iff \forall\, y \in R_2,\quad (x,y) \in R_1 \\ &\iff \text{non}\left(\exists\, y \in R_2,\quad (x,y) \notin R_1\right) \\ &\iff \nexists\, y \in R_2,\quad (x,y) \notin R_1 \end{aligned}$$

- Avec deux `NOT EXISTS` imbriqués :

$$\begin{aligned} x \in R_1 \div R_2 &\iff \forall\, y \in R_2,\quad (x,y) \in R_1 \\ &\iff \nexists y \in R_2 \mid \nexists z \in R_1 \;;\; z = (x,y) \end{aligned}$$

- Avec `COUNT` : $x \in R_1 \div R_2$ si et seulement si le nombre de $(x,y) \in R_1$ tel que $y \in R_2$ est le nombre de tuples de R_2.

Exemples

E1 – Numéros des professeurs enseignant à tous les étudiants

- il n'existe pas d'étudiants qui ne soit pas dans l'une des classes de ce professeur.

```
SELECT num_p FROM prof
WHERE NOT EXISTS (SELECT * FROM eleve e
                  WHERE num_classe NOT IN
                          (SELECT num_classe FROM enseigne
                           WHERE num_prof = num_p)
                  );
```

- il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant.

```
SELECT num_p FROM prof
WHERE NOT EXISTS
        (SELECT * FROM eleve e
         WHERE NOT EXISTS (SELECT * FROM enseigne
                           WHERE num_prof = num_p
                             AND num_classe = e.num_classe)
         );
```

- Le nombre d'étudiant qui suivent un cours de ce professeur est le nombre total d'étudiants.

```
SELECT num_p FROM prof
WHERE (SELECT COUNT(*) FROM eleve e)
      = (SELECT COUNT(DISTINCT *) FROM eleve e
         JOIN enseigne en ON e.num_classe = enseigne.num_classe
         WHERE en.num_prof = num_p );
```

```
SELECT num_prof
FROM enseigne JOIN classe c ON num_classe = no
              JOIN eleve e  ON e.num_classe = c.num_classe
GROUP BY num_prof
HAVING COUNT(DISTINCT e.num_p) = (SELECT COUNT(*) FROM eleve);
```

E2 – Nom des fournisseurs qui fournissent tous les produits rouges

- il n'existe pas de produit rouge qui ne soit pas dans les livraisons par ce fournisseur de ce produit.

```
SELECT nomF FROM F
WHERE NOT EXISTS (SELECT * FROM P
                  WHERE couleur = 'rouge'
                  AND nP NOT IN (SELECT np FROM L
                                 WHERE nF = F.nF)
                  );
```

- il n'existe pas de produit rouge tel qu'il n'existe pas de livraison par ce fournisseur de ce produit.

```
SELECT nomF FROM F
WHERE NOT EXISTS (SELECT * FROM P
                  WHERE couleur = 'rouge'
                  AND NOT EXISTS (SELECT * FROM L
                                  WHERE nF = F.nF
                                    AND nP = P.nP)
                  );
```

- le nombre de produits rouges est égal au nombre de produits rouges livrés par ce fournisseur.

```
SELECT nomF FROM F
WHERE (SELECT COUNT(*) FROM P WHERE couleur = 'rouge')
      = (SELECT COUNT(DISTINCT L.nP)
         FROM L JOIN P ON L.nP = P.nP
         WHERE nF = F.nF AND couleur = 'rouge'
        );
```

```
SELECT nF
FROM L JOIN P ON L.nP = P.nP
WHERE couleur = 'rouge'
GROUP BY nF
HAVING COUNT(DISTINCT nP) = (SELECT COUNT(*) FROM P
                             WHERE couleur = 'rouge');
```