Complément de SQL - Représentations graphiques

LYCÉE CARNOT (DIJON), 2019 - 2020

Germain Gondor

Sommaire

Sommaire

Relation d'ordre

La clause `ORDER BY` permet de trier les résultats. Par défaut l'ordre est ascendant `ASC` mais avec `DESC` on peut spécifier l'ordre descendant du tri.

Relation d'ordre

La clause `ORDER BY` permet de trier les résultats. Par défaut l'ordre est ascendant `ASC` mais avec `DESC` on peut spécifier l'ordre descendant du tri.

Quand plusieurs attributs sont sélectionnés, le tri des colonnes se fait d'abord selon la colonne du premier attribut, puis selon les suivantes en cas d'égalité des premières.

Relation d'ordre

La clause `ORDER BY` permet de trier les résultats. Par défaut l'ordre est ascendant `ASC` mais avec `DESC` on peut spécifier l'ordre descendant du tri.

Quand plusieurs attributs sont sélectionnés, le tri des colonnes se fait d'abord selon la colonne du premier attribut, puis selon les suivantes en cas d'égalité des premières.

```
SELECT * FROM R1 ORDER BY att1 DESC, att2 ASC;
```

Relation d'ordre

EXEMPLE : faire apparaître les départements triés ordre décroissant dans chaque département les numéros de station triés par ordre croissant :

```
SELECT departement,num_sta
FROM radome
ORDER BY departement DESC, num_sta ASC;
```

Relation d'ordre

EXEMPLE : faire apparaître les départements triés ordre décroissant dans chaque département les numéros de station triés par ordre croissant :

```
SELECT departement,num_sta
FROM radome
ORDER BY departement DESC, num_sta ASC;
```

La clause `ORDER BY` se positionne en fin de requête SQL puisqu'elle permet de trier le résultat final et doit figurer d'une seule fois après la clause `SELECT`.

Sous-requête renvoyant un unique élément

Grâce aux fonctions d'agrégation, nous avons vu (cours précédent) qu'il était possible d'effectuer des opérations qui, sur l'ensemble des valeurs d'un attribut d'une relation, renvoient un résultat composé d'un unique élément.

Sous-requête renvoyant un unique élément

Grâce aux fonctions d'agrégation, nous avons vu (cours précédent) qu'il était possible d'effectuer des opérations qui, sur l'ensemble des valeurs d'un attribut d'une relation, renvoient un résultat composé d'un unique élément.

Il est alors possible d'utiliser ce résultat dans une sélection :

```
SELECT ... FROM ... WHERE truc = (SELECT ... FROM ....) ;
```

Sous-requête renvoyant un unique élément

EXEMPLE : déterminer le nom et l'altitude de la plus haute station météo de la relation `RADOME` :

```
SELECT nom_usuel, altitude FROM radome
WHERE altitude=(SELECT max(altitude) FROM radome);
```

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$
- `op ANY` c'est l'existence $\exists$ associée à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$
- `op ANY` c'est l'existence $\exists$ associée à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<\ldots$
- `op ALL` c'est le quelque soit $\forall$ associé à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<\ldots$

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$
- `op ANY` c'est l'existence $\exists$ associée à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...
- `op ALL` c'est le quelque soit $\forall$ associé à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...

REMARQUES:

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$
- `op ANY` c'est l'existence $\exists$ associée à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...
- `op ALL` c'est le quelque soit $\forall$ associé à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...

REMARQUES:

- `IN` produit le même effet que `= ANY`

Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$
- `op ANY` c'est l'existence $\exists$ associée à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...
- `op ALL` c'est le quelque soit $\forall$ associé à un opérateur `op` où `op` est un opérateur comme par exemple $=$, $!=$, $<$...

REMARQUES:

- `IN` produit le même effet que `= ANY`
- `NOT IN` produit le même effet que `!= ALL`

Sous-requête renvoyant plusieurs éléments

EXEMPLE : quelles sont les noms de villes dont la température était supérieure à 297 K le jour de la mesure ?

Sous-requête renvoyant plusieurs éléments

EXEMPLE : quelles sont les noms de villes dont la température était supérieure à 297 K le jour de la mesure ?

```
SELECT nom_usuel FROM radome
WHERE id_omm IN
          (SELECT id_omm FROM synop WHERE t>295);
```

Sous-requête renvoyant plusieurs éléments

EXEMPLE : quelles sont les noms de villes dont la température était supérieure à 297 K le jour de la mesure ?

```
SELECT nom_usuel FROM radome
WHERE id_omm IN
          (SELECT id_omm FROM synop WHERE t>295);
```

qui donne le même résultat que :

```
SELECT nom_usuel FROM radome ra JOIN synop sy
ON ra.id_omm=sy.id_omm WHERE t>295;
```

Sous-requête renvoyant plusieurs éléments

EXEMPLE : quelles sont les villes non situées en Bourgogne ?

```
SELECT nom_usuel  FROM radome
        WHERE departement NOT IN (21,58,71,89);
```

```
SELECT nom_usuel  FROM radome
WHERE departement NOT IN
        (SELECT departement FROM radome
                WHERE region="Bourgogne");
```

Cas de l'existence

La requête `EXISTS sous-req` donne un résultat vrai si la sous requête `sous-req` retourne au moins une ligne.

```
SELECT ... FROM ... WHERE EXISTS (SELECT ...);
```

Par ailleurs, la requête `NOT EXISTS sous-req` donne un résultat vrai si la sous requête `sous-req` ne retourne pas de ligne.

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

Cas de l'existence

EXEMPLE : quels sont les noms des élèves n'étant pas seuls issus d'un lycée ?

```sql
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

```sql
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);
```

Cas de l'existence

ou encore avec deux jointures :

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

Cas de la division

Nous avons vu dans le cours précédent que le division ne fait pas actuellement partie de la norme SQL. Cependant, comment répondre à la question :

Cas de la division

Nous avons vu dans le cours précédent que le division ne fait pas actuellement partie de la norme SQL. Cependant, comment répondre à la question :

PROBLÉMATIQUE : Quels sont les numéros des professeurs enseignant à tous les étudiants ?

Cas de la division

Nous avons vu dans le cours précédent que le division ne fait pas actuellement partie de la norme SQL. Cependant, comment répondre à la question :

PROBLÉMATIQUE : Quels sont les numéros des professeurs enseignant à tous les étudiants ?

On peut reformuler le problème sous la forme :

Cas de la division

Nous avons vu dans le cours précédent que le division ne fait pas actuellement partie de la norme SQL. Cependant, comment répondre à la question :

PROBLÉMATIQUE : Quels sont les numéros des professeurs enseignant à tous les étudiants ?

On peut reformuler le problème sous la forme :

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels que quelque soit l'étudiant, cet étudiant est dans une des classes du professeur ?

Hélas, nouveau problème, si `EXISTS` ($\exists$) existe , `FORALL` ($\forall$) n'existe pas ! On s'en sort alors avec deux `NOT EXIST` imbriqués.

Cas de la division

Nous avons vu dans le cours précédent que le division ne fait pas actuellement partie de la norme SQL. Cependant, comment répondre à la question :

PROBLÉMATIQUE : Quels sont les numéros des professeurs enseignant à tous les étudiants ?

On peut reformuler le problème sous la forme :

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels que quelque soit l'étudiant, cet étudiant est dans une des classes du professeur ?

Hélas, nouveau problème, si `EXISTS` ($\exists$) existe , `FORALL` ($\forall$) n'existe pas ! On s'en sort alors avec deux `NOT EXIST` imbriqués.En effet : $t \in R_1 \div R_2 \Leftrightarrow \forall t' \in R_2, (t, t') \in R_1 \Leftrightarrow \nexists t' \in R_2 \mid \nexists t'' \in R_1 \ ; \ t'' = (t, t')$.

Cas de la division

PROBLÉMATIQUE :

Cas de la division

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant ?

Cas de la division

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant ?

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

Cas de la division

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant ?

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

Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants qui ne soit pas dans l'une des classes de ce professeur ?

Cas de la division

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant ?

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

Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants qui ne soit pas dans l'une des classes de ce professeur ?

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

Cas de la division

ce qui pourra se faire de façon simple avec :

Cas de la division

ce qui pourra se faire de façon simple avec :

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels que le nombre d'étudiant qui suivent un cours de ce professeur est le nombre total d'étudiants.

Cas de la division

ce qui pourra se faire de façon simple avec :

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels que 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 (*)
FROM eleve e JOIN enseigne en ON e.num_classe = en.num_clas
        WHERE en.num_prof = num_p );
```

Regroupement par attributs communs

OBJECTIF : La clause `GROUP BY` permet de regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat. Elle ne renvoie qu'une seule ligne par groupe.

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

Regroupement par attributs communs

OBJECTIF : La clause `GROUP BY` permet de regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat. Elle ne renvoie qu'une seule ligne par groupe.

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

ATTENTION ! il faut que le contenu du `SELECT` soit cohérent, i.e. :

Regroupement par attributs communs

OBJECTIF : La clause `GROUP BY` permet de regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat. Elle ne renvoie qu'une seule ligne par groupe.

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

ATTENTION ! il faut que le contenu du `SELECT` soit cohérent, i.e. :

- soit composé des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe

Regroupement par attributs communs

OBJECTIF : La clause `GROUP BY` permet de regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat. Elle ne renvoie qu'une seule ligne par groupe.

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

ATTENTION ! il faut que le contenu du `SELECT` soit cohérent, i.e. :

- soit composé des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe
- soit des fonctions d'agrégat (qui porteront sur chaque groupe)

Regroupement par attributs communs

EXEMPLE : Quels sont les nombres de cours dispensés par chaque enseignant ?

```
SELECT nom,prénom, COUNT (*)
FROM enseigne en JOIN personne pe ON en.num_prof=pe.num_p
GROUP BY nom,prénom;
```

Regroupement par attributs communs

EXEMPLE : Quel est le nombre moyen d'élèves pour chaque professeur ?

```
1 SELECT nom,"Effectif moyen" FROM personne pe JOIN
2         (SELECT num_prof, AVG(effectif) AS "Effectif moyen"
3                 FROM classe JOIN enseigne ON num_classe = no
4                 GROUP BY num_prof) gr
5         ON pe.num_p=gr.num_prof;
```

Regroupement par attributs communs

ATTENTION ! si des professeurs n'ont pas d'élèves, ils n'apparaîtront pas. On peut les ajouter avec une `UNION`.

```
1 SELECT nom,"Effectif moyen" FROM personne pe JOIN
2         (SELECT num_p, 0 AS "Effectif moyen" FROM prof
3                 WHERE num_p NOT IN (SELECT num_prof from enseigne)
4         UNION
5         SELECT num_prof, AVG(effectif) AS "Effectif moyen"
6                 FROM classe JOIN enseigne ON num_classe = no
7                 GROUP BY num_prof) gr
8         ON pe.num_p=gr.num_p;
```

Conditions sur les regroupements par attributs communs

OBJECTIF : La clause `HAVING` permet d'imposer une condition sur les groupes définis par `GROUP BY`. C'est l'équivalent de la clause `WHERE` après la clause `SELECT`. C'est pour cela qu'elle se place après `GROUP BY` :

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

Conditions sur les regroupements par attributs communs

REMARQUE: Le prédicat dans la clause `HAVING` suit les mêmes règles de syntaxe qu'un prédicat figurant dans une clause `WHERE`. Cependant, il ne peut porter que sur les caractéristiques du groupe :

- soit sur les fonctions d'agrégation (mais pas nécessairement celles qui figurent dans la clause `WHERE`)
- soit sur des expressions (généralement des colonnes) figurant dans la clause `GROUP BY`.

Conditions sur les regroupements par attributs communs

EXEMPLE : Quels sont les 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 person
GROUP BY num_prof HAVING COUNT(DISTINCT num_classe) > 1;
```

Agrégation sur deux niveaux

La clause `GROUP BY` renvoyant plusieurs résultats, on peut appliquer une fonction d'agrégation sur une sélection obtenue avec un regroupement.

EXEMPLE : Quelle est la moyenne du nombre total d'élèves par professeur (ayant des élèves...).

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

Bloc de qualification

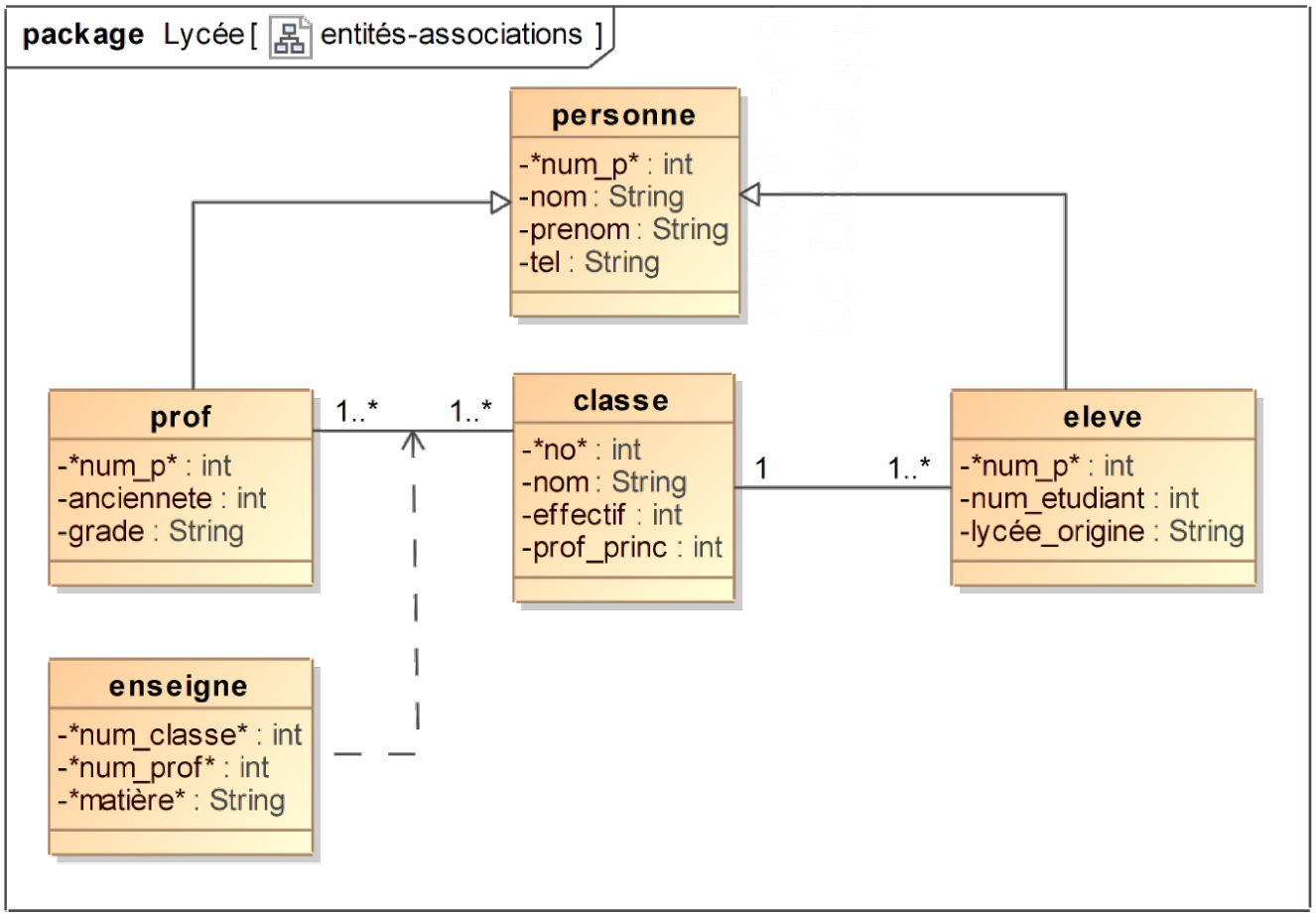
Après avoir vu les principales clauses d'une requête SQL, la structure de base est un bloc de qualification où la syntaxe impose l'ordre suivant :

```
1 SELECT Ai,..., An          -- colonnes et agregations
2 FROM R                     -- relation
3 WHERE F                    -- assertion
4 GROUP BY A                 -- regroupement
5 HAVING H                   -- assertion
6 ORDER BY T                 -- tri
7 ;
```

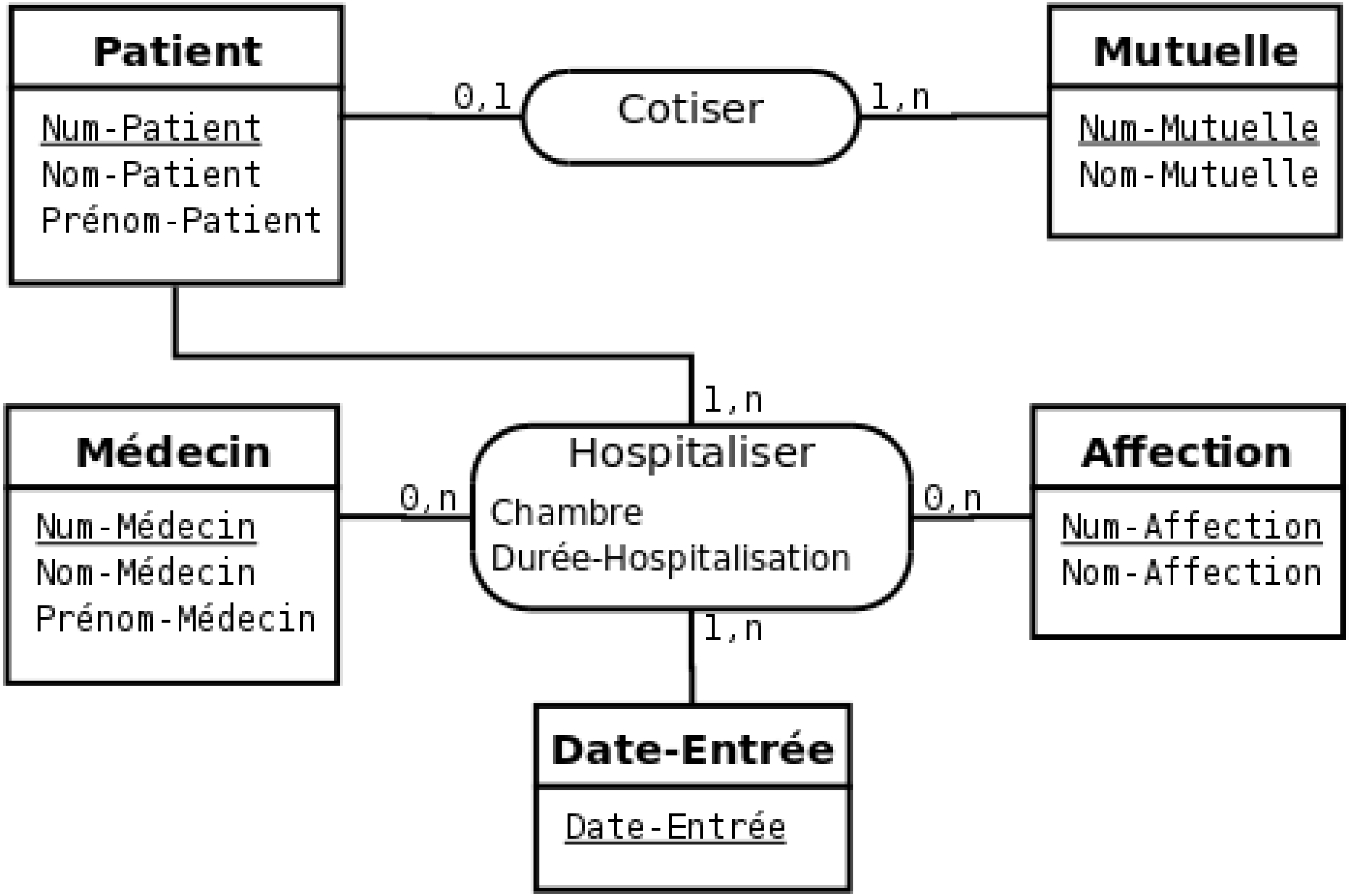
Sommaire

Modèle entités-associations

A l'aide d'un diagramme de classe d'UML, un schéma relationnel appelé diagramme entités-associations peut être représenté :



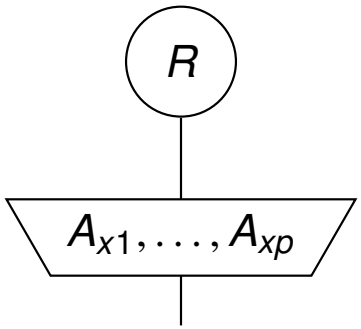
Modèle entités-associations



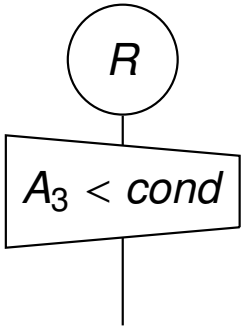
Éléments graphiques associés à l'algèbre relationnel

Pour raisonner sur le papier ou traduire visuellement une requête voici quelques éléments de langage graphique :

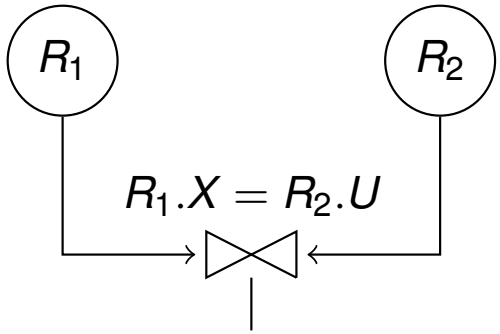
Projection



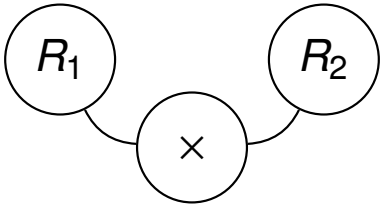
Selection



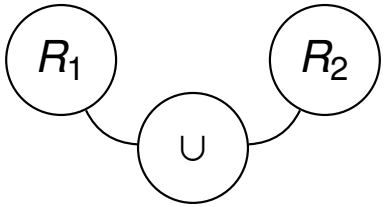
Jointure



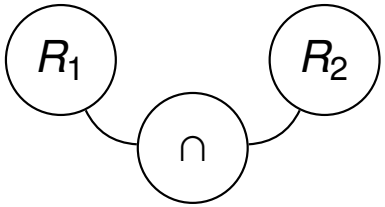
Produit cartésien



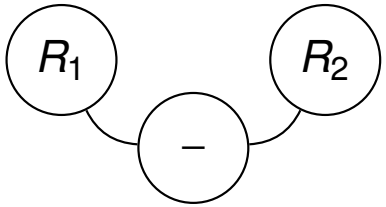
Union



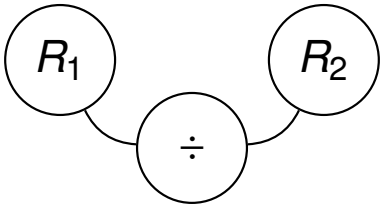
Intersection



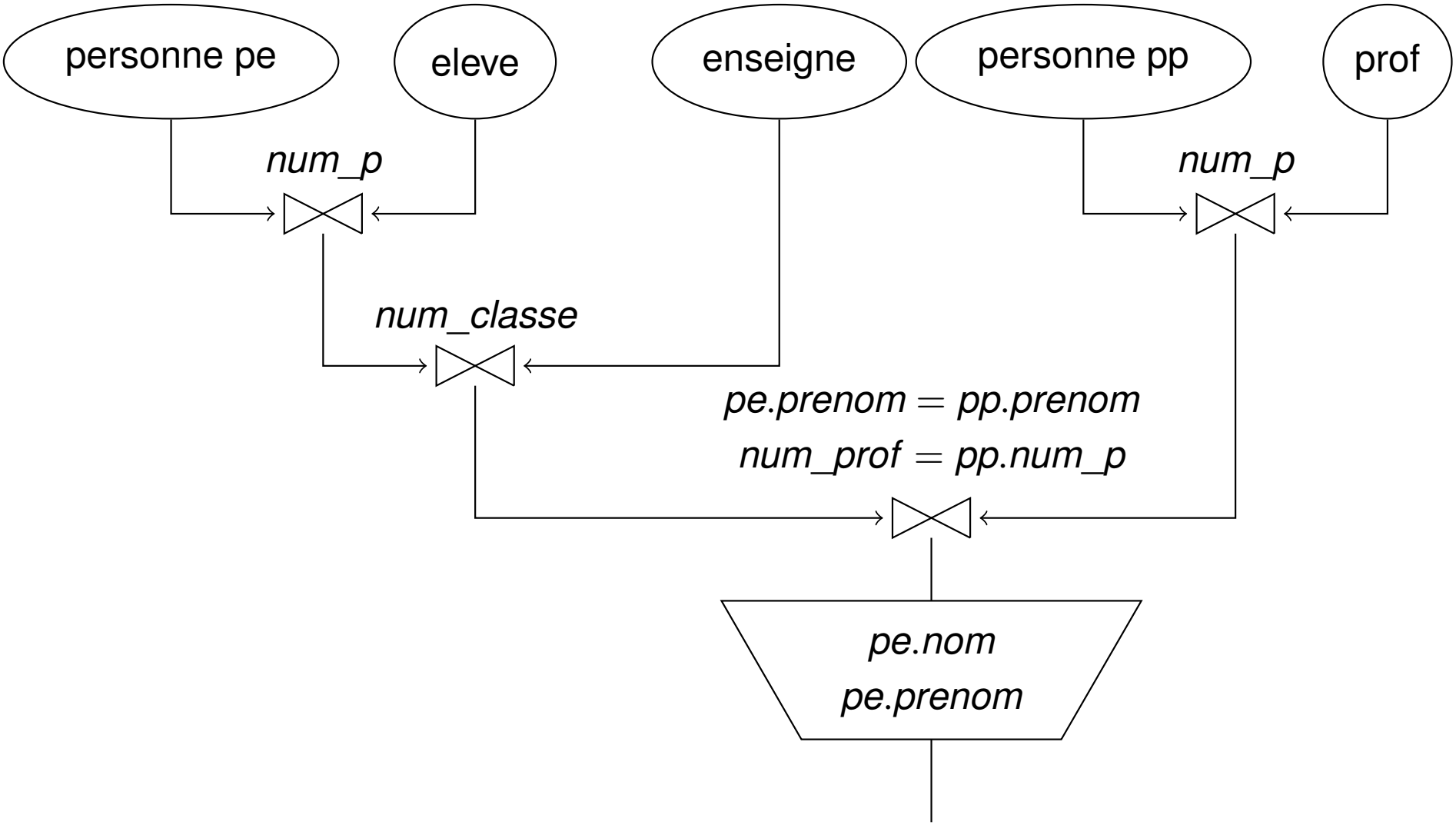
Différence

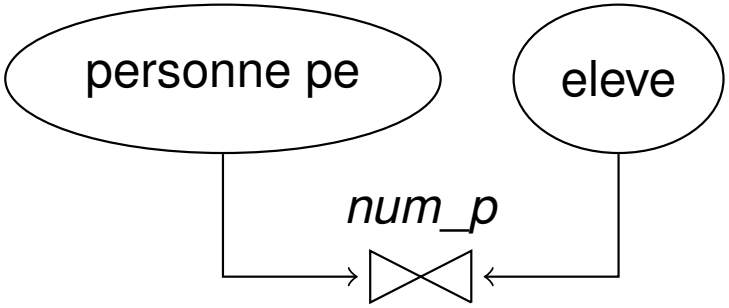


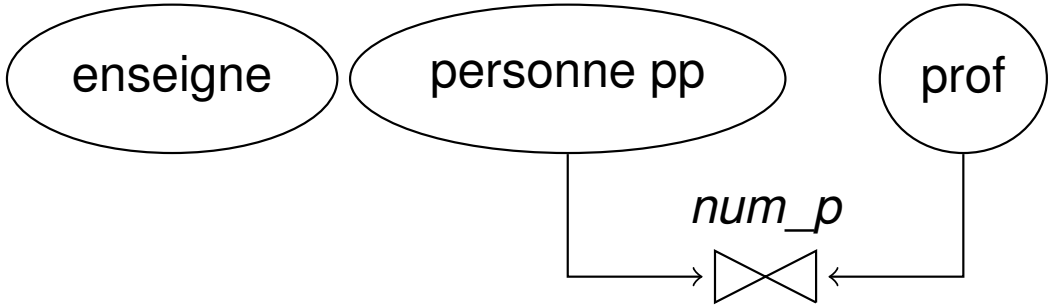
Division

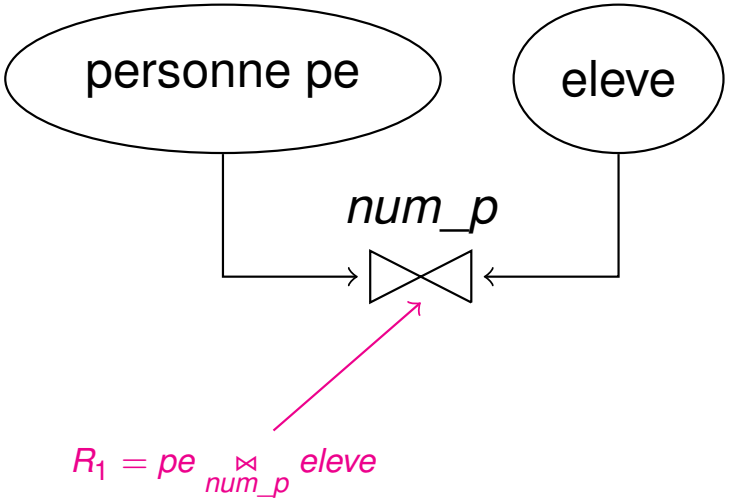


Éléments graphiques associés à l'algèbre relationnel

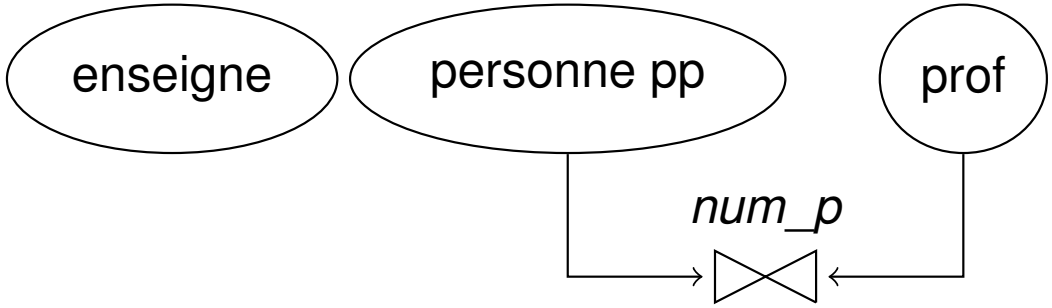


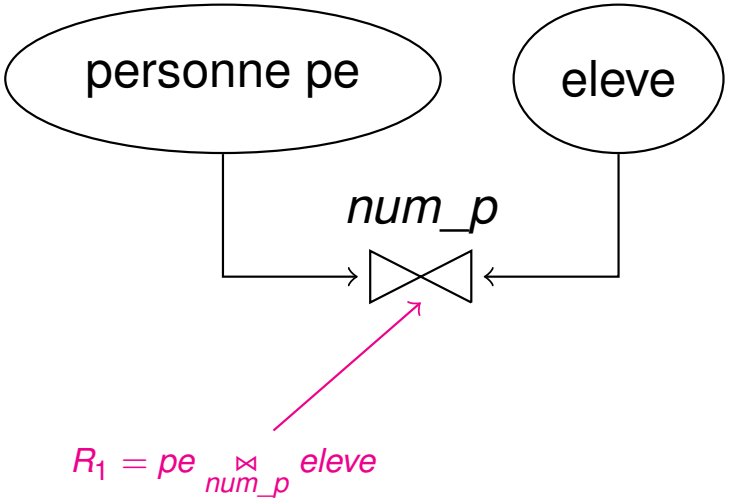

personne pe
eleve
num_p

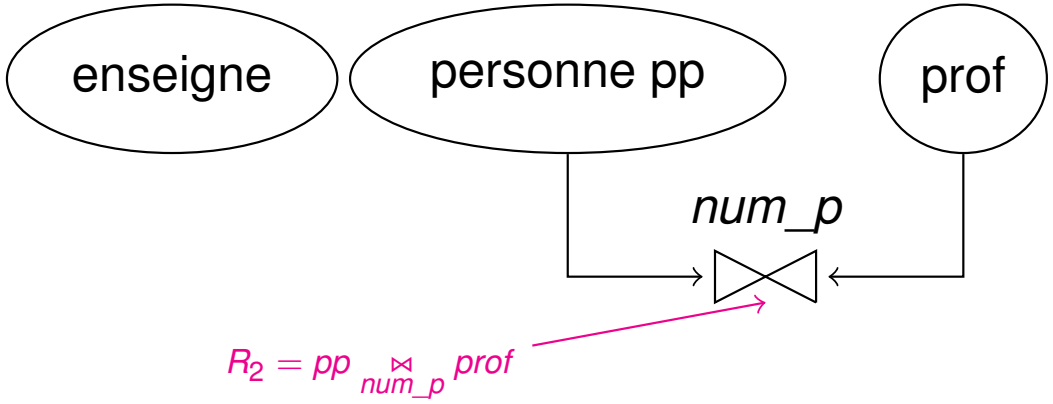

enseigne
personne pp
prof
num_p

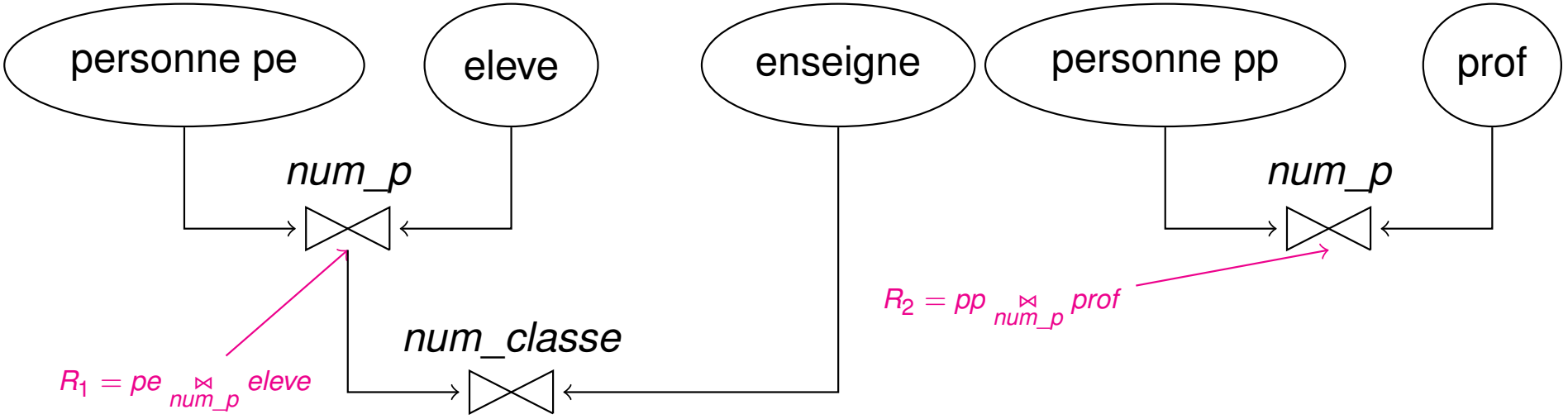


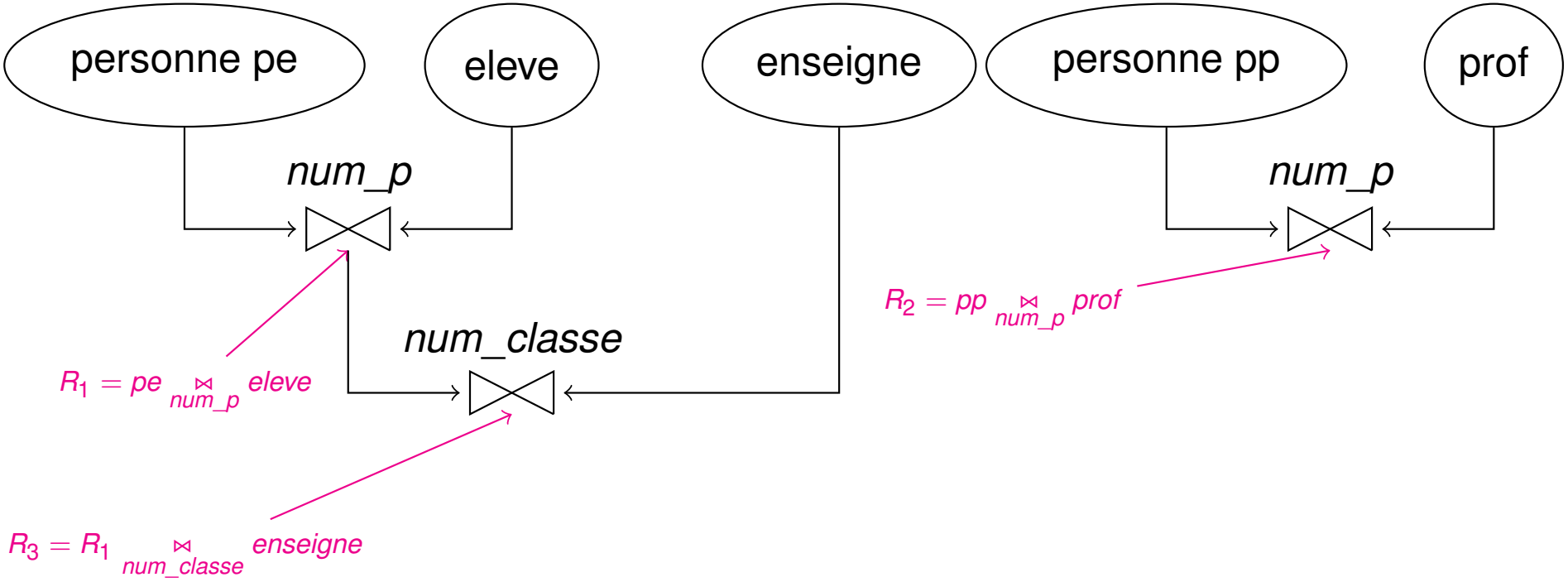
$R_1 = pe \underset{num\_p}{\bowtie} eleve$

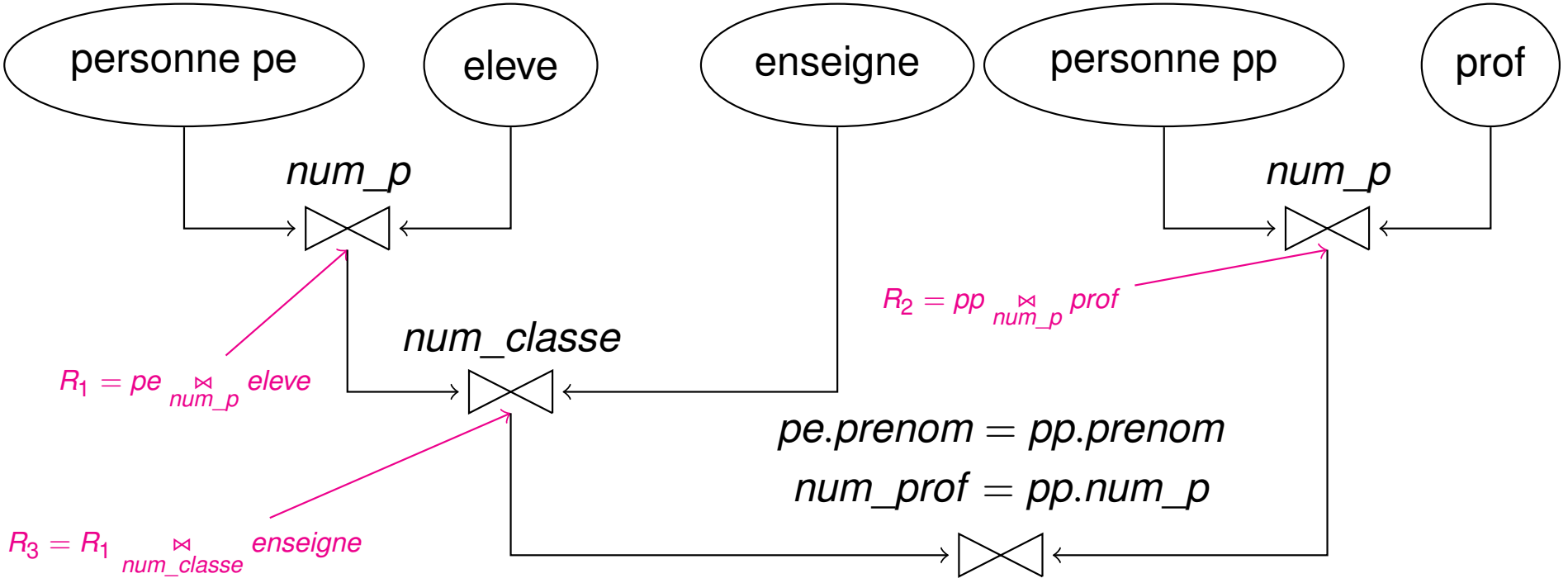


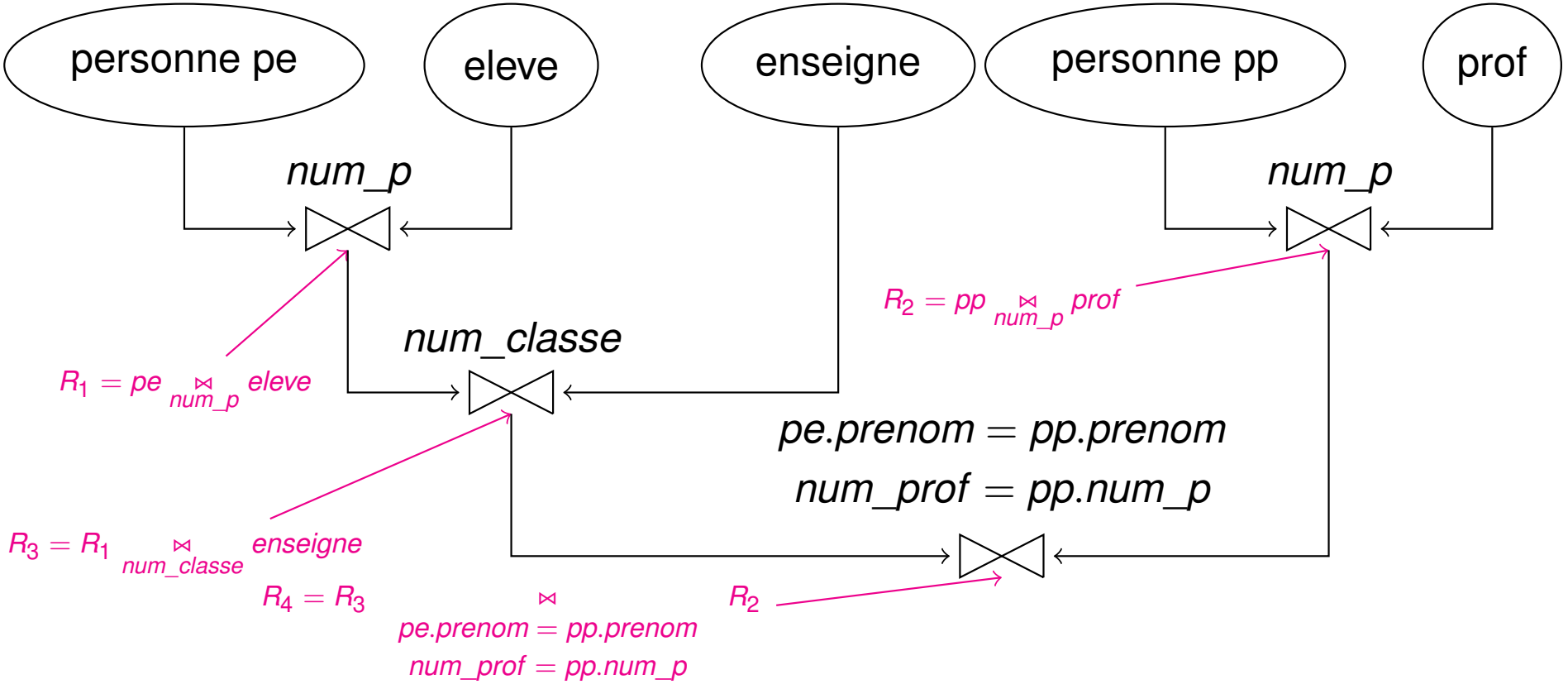

personne pe
eleve
num_p
$R_1 = pe \underset{num\_p}{\bowtie} eleve$

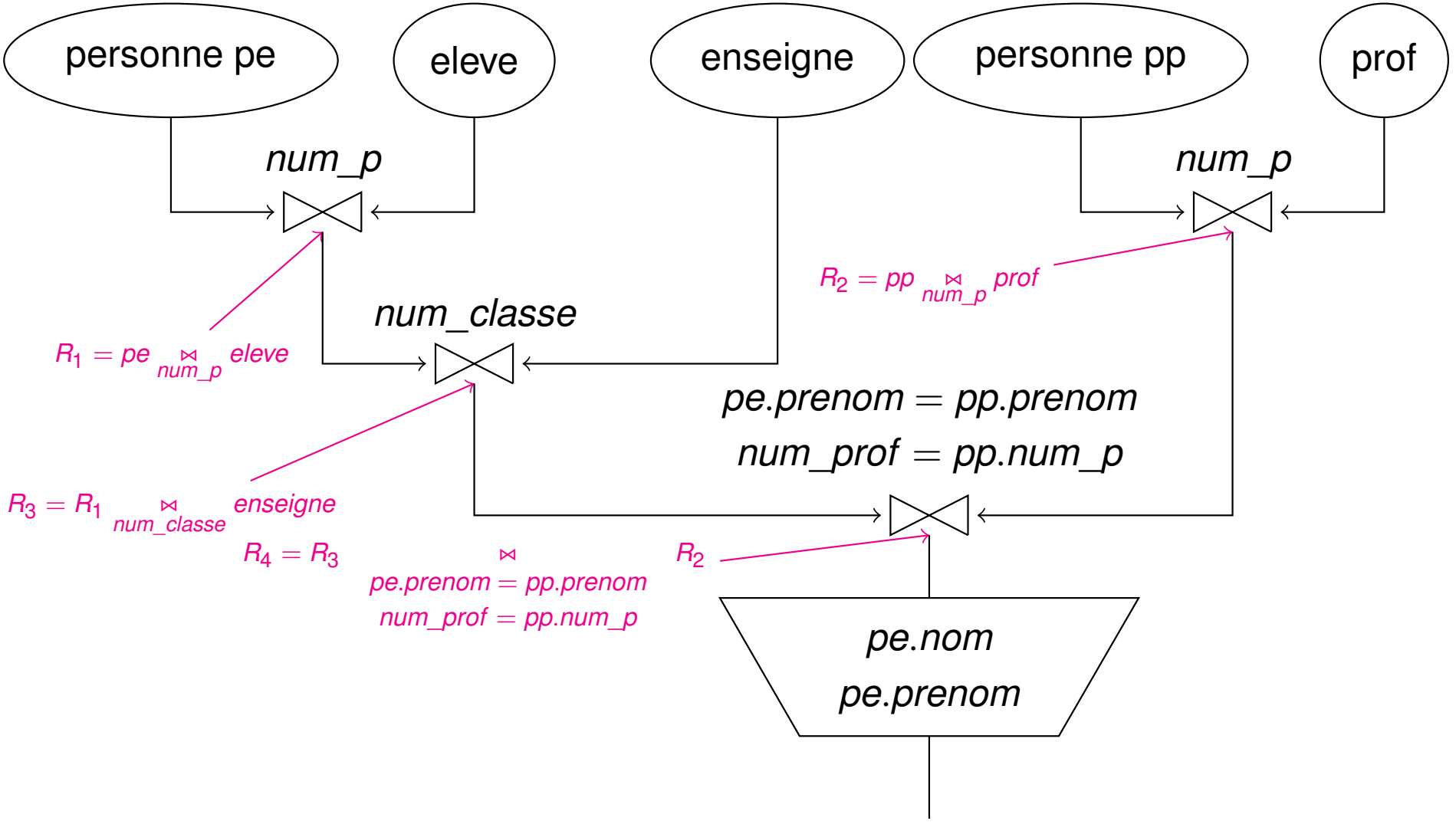

enseigne
personne pp
prof
num_p
$R_2 = pp \underset{num\_p}{\bowtie} prof$

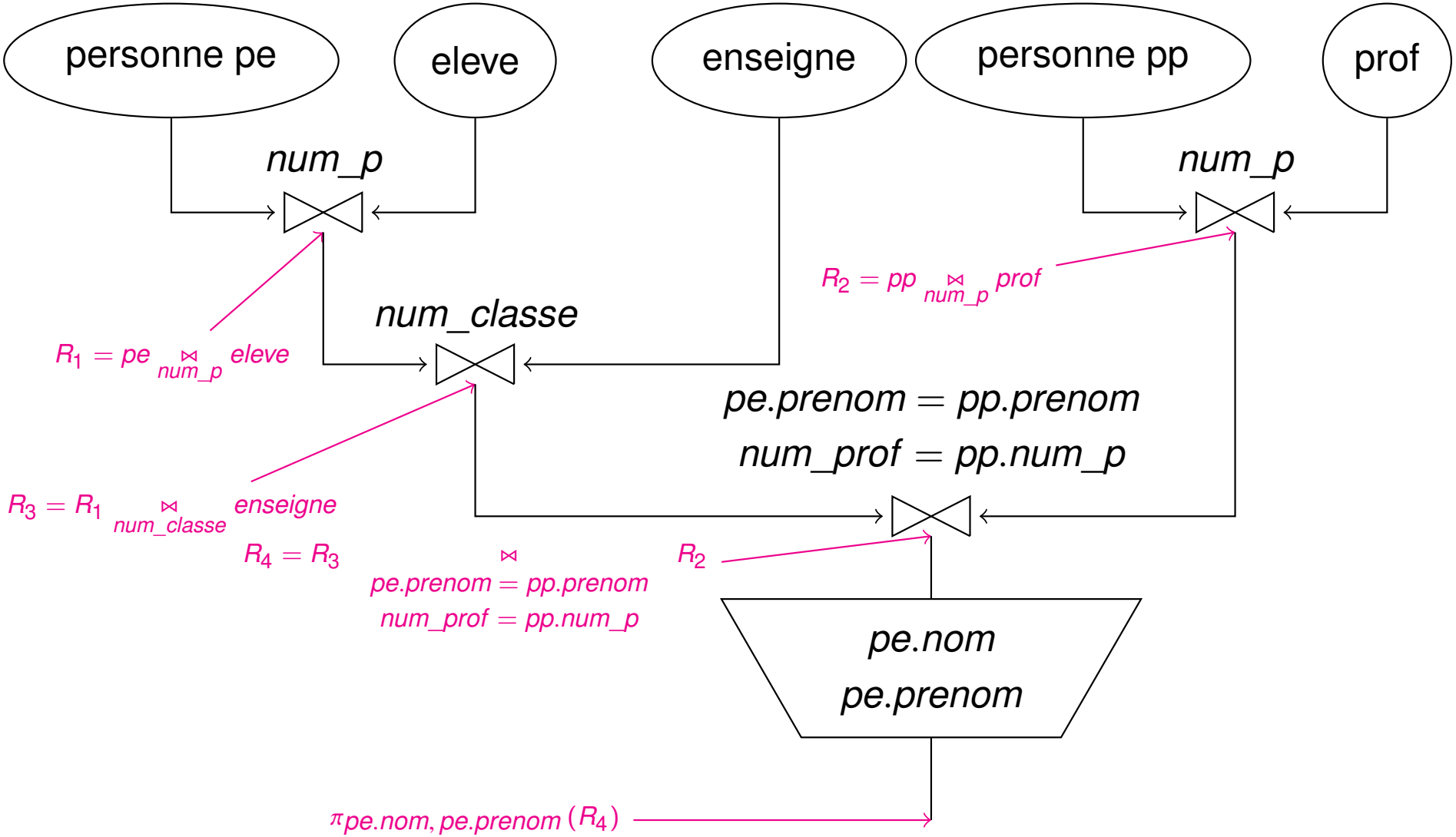

personne pe
eleve
enseigne
personne pp
prof
num_p
num_p
num_classe
$R_1 = pe \underset{num\_p}{\bowtie} eleve$
$R_2 = pp \underset{num\_p}{\bowtie} prof$

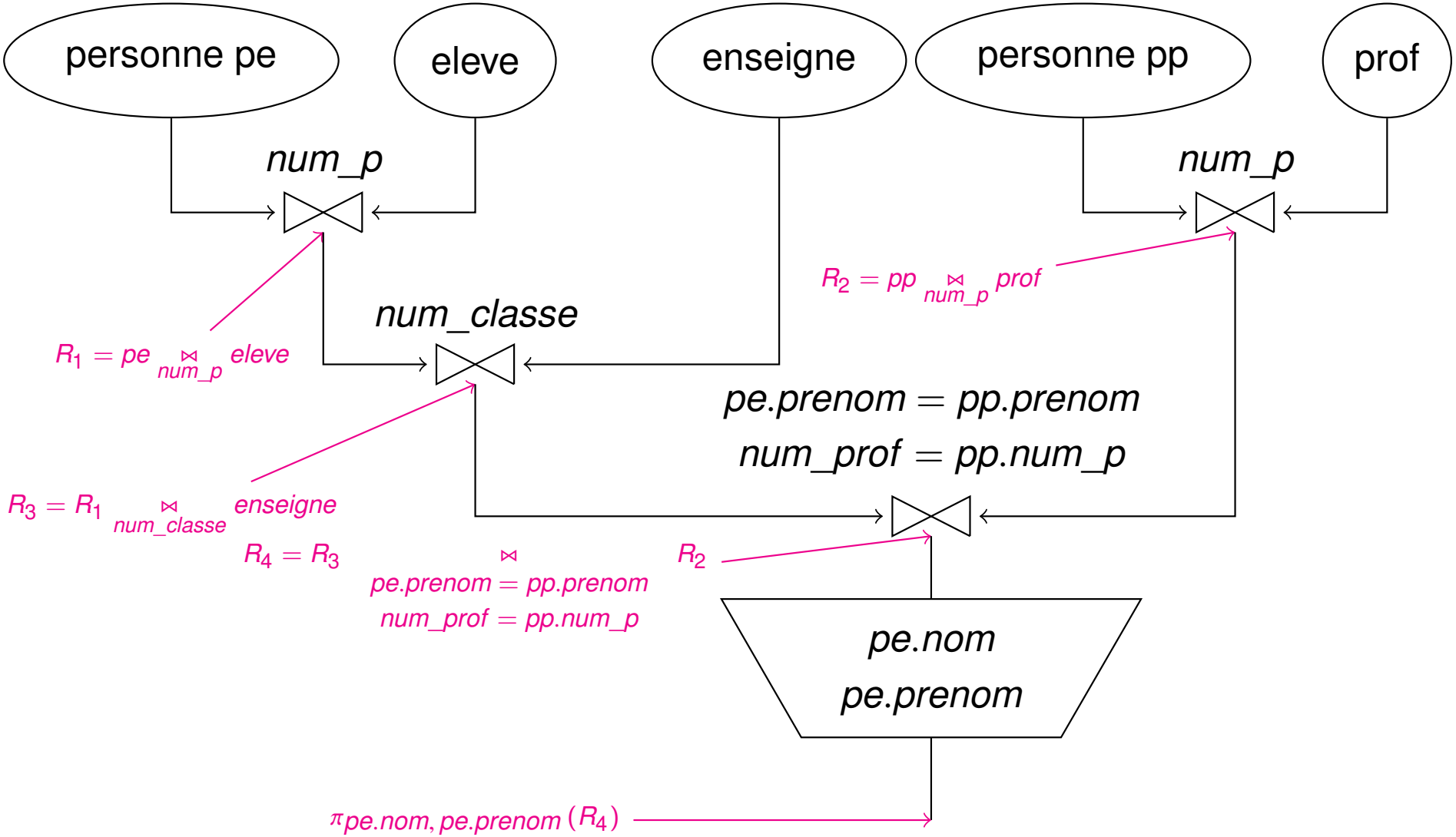

personne pe
eleve
enseigne
personne pp
prof
num_p
num_p
num_classe
$R_1 = pe \underset{num\_p}{\bowtie} eleve$
$R_2 = pp \underset{num\_p}{\bowtie} prof$
$R_3 = R_1 \underset{num\_classe}{\bowtie} enseigne$


personne pe
eleve
enseigne
personne pp
prof
num_p
num_p
num_classe
$R_1 = pe \underset{num\_p}{\bowtie} eleve$
$R_2 = pp \underset{num\_p}{\bowtie} prof$
$pe.prenom = pp.prenom$
$num\_prof = pp.num\_p$
$R_3 = R_1 \underset{num\_classe}{\bowtie} enseigne$


personne pe
eleve
enseigne
personne pp
prof
num_p
num_p
num_classe
$R_1 = pe \underset{num\_p}{\bowtie} eleve$
$R_2 = pp \underset{num\_p}{\bowtie} prof$
$pe.prenom = pp.prenom$
$num\_prof = pp.num\_p$
$R_3 = R_1 \underset{num\_classe}{\bowtie} enseigne$
$R_4 = R_3 \underset{pe.prenom = pp.prenom \; num\_prof = pp.num\_p}{\bowtie} R_2$


personne pe
eleve
enseigne
personne pp
prof
num_p
num_p
num_classe
$R_1 = pe \underset{num\_p}{\bowtie} eleve$
$R_2 = pp \underset{num\_p}{\bowtie} prof$
$R_3 = R_1 \underset{num\_classe}{\bowtie} enseigne$
$R_4 = R_3 \underset{pe.prenom = pp.prenom \; num\_prof = pp.num\_p}{\bowtie} R_2$
pe.prenom = pp.prenom
num_prof = pp.num_p
pe.nom
pe.prenom


personne pe
eleve
enseigne
personne pp
prof
num_p
num_p
num_classe
$R_1 = pe \underset{num\_p}{\bowtie} eleve$
$R_2 = pp \underset{num\_p}{\bowtie} prof$
$pe.prenom = pp.prenom$
$num\_prof = pp.num\_p$
$R_3 = R_1 \underset{num\_classe}{\bowtie} enseigne$
$R_4 = R_3 \underset{pe.prenom = pp.prenom \; num\_prof = pp.num\_p}{\bowtie} R_2$
pe.nom
pe.prenom
$\pi_{pe.nom, pe.prenom}(R_4)$



Quels sont les nom et prénom des élèves ayant le même prénom qu'un de leurs professeurs ?

```
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
JOIN prof p ON p.num_p = pp.num_p;
```