UNIVERSITÉ DE LORRAINE | IUT Metz

Bases de Données : SQL avancé

Bernard HEULLUY
bernard.heulluy@univ-lorraine.fr

- **Volume** : 5 heures cours magistral – 8 heures de TD – 4 heures de TP
- **Contrôle des connaissances :** Projet, contrôle écrit en fin de semestre

  modalités contrôle : aucun document sauf 1 feuille A4 recto simple manuscrite

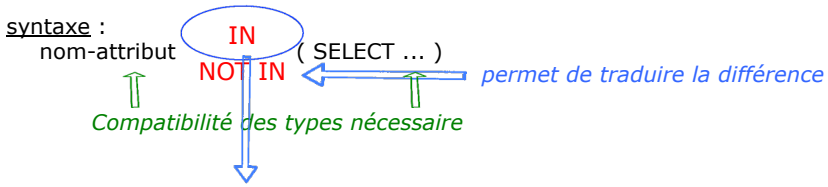
Chapitre 1 : AUTRE TRAITEMENT DE LA JOINTURE

I- Traduction de la jointure en SQL avec IN

1- Référence de départ : S1 - Chapitre 3 - Paragraphe IX : Différence (3e forme de sous condition dans une clause WHERE)

- **3-** sous-conditions avec sous-interrogation

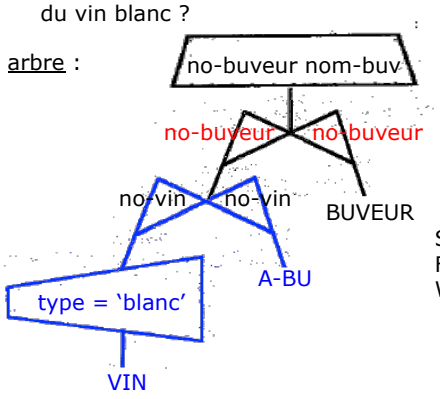
  3-1- vérifier l'appartenance (ou la non appartenance) de la valeur d'un attribut au résultat d'une sous-interrogation SELECT



AUTRE FORME DE LA JOINTURE

✓ **Exemple - Requête 2-1** : quels sont les buveurs qui ont déjà acheté du vin blanc ?

arbre :



Traduction en SQL :

```
SELECT no-buveur, nom-buv
FROM buveur
WHERE no-buveur IN
      ( SELECT no-buveur
        FROM vin, a-bu
        WHERE vin.no-vin = a-bu.no-vin
          AND type = 'blanc' )
```

Littéralement :

Liste des buveurs dont le numéro fait partie de l'ensemble des numéros des buveurs ayant déjà acheté du vin blanc

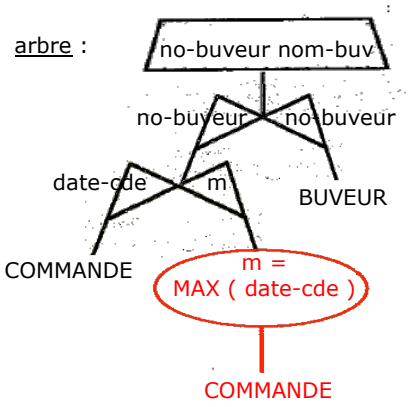
Remarques : *différences par rapport au schéma de traduction classique :*

- *les attributs figurant dans le résultat ne peuvent être issus de la sous-interrogation ;*
- *sinon équivalence sauf dans certains cas particuliers, notamment les jointures avec le résultat d'une opération arithmétique.*

II- Cas particulier

✓ **Exemples - Requête 2-2** : quel(s) buveur(s) a passé la dernière commande enregistrée ?

arbre :



Traduction en SQL :

```
SELECT no-buveur, nom-buv
FROM buveur, commande
WHERE commande.no-buveur = buveur.no-buveur
   AND date-cde IN
      ( SELECT MAX ( date-cde )
        FROM commande )
```

Alternative :

Sauvegarder le résultat de l'opérateur MAX dans une relation temporaire.

Traduction alternative en SQL :

```
SELECT no-buveur, nom-buv
FROM buveur, commande
WHERE commande.no-buveur = buveur.no-buveur
   AND date-cde IN
      ( SELECT MAX ( date-cde )
        FROM commande )
```

```
CREATE TEMPORARY TABLE reltemp
SELECT MAX ( date-cde ) m
FROM commande
;
SELECT no-buveur, nom-buv
FROM buveur, commande, reltemp
WHERE commande.no-buveur = buveur.no-buveur
   AND date-cde = m
;
DROP TABLE reltemp
```

Remarque :
Dans certains dialecte SQL, IN peut dans ce contexte être remplacé par =.

Chapitre 2 : LA PARTITION

I- Principe et définition

✓ **Objectifs :**

1 décomposer une relation en un certain nombre de sous-relations de même schéma et vérifiant chacune certaines propriétés ;

2 éventuellement, exprimer une condition de sélection portant sur ces sous-relations ;

3 produire 1 n-uplet résultat par sous-relation sélectionnée.

-> 1 Partition d'une relation R ; soit R une relation et K un sous-ensemble non vide d'attributs de R ; la partition de R suivant K, notée Part (R/K), est définie par :

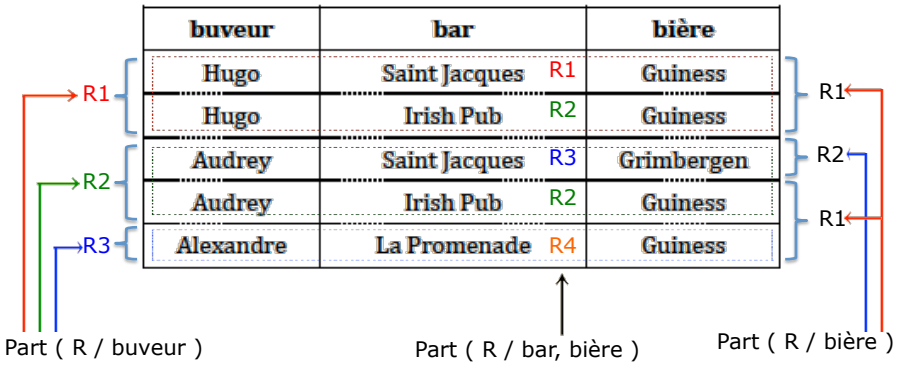
$$\text{Part}(R/K) = \{ R_i / \bigcup_i R_i = R \text{ et } (R_i \cap R_j = \emptyset \text{ si } i \neq j) \text{ et cardinal } (R_i[K]) = 1 \text{ et } (R_i[K] \neq R_j[K] \text{ si } i \neq j) \}$$

=> La relation R est décomposée en sous-relations de même schéma que R et dont, pour chacune, tous les n-uplets ont la même valeur pour l'ensemble d'attributs K.

-> 2 Condition exprimée à l'aide d'opérateurs arithmétiques appliqués aux sous-relations ;

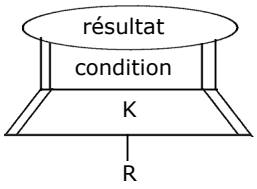
-> 3 Ce n-uplet est composé d'attributs de K et/ou d'opérations arithmétiques appliquées à la sous-relation concernée.

✓ **Exemple** : R (buveur, bar, bière)

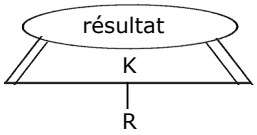


II- Notation

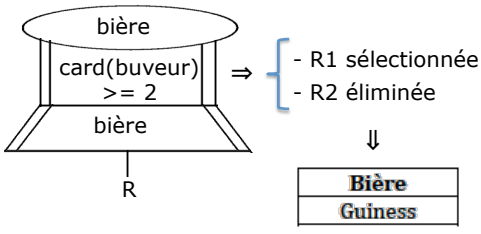
- **Partition avec sélection des sous-relations :**



- **Partition sans sélection des sous-relations :**



- <u>Exemple simple</u> :
  Liste des bières bues par au moins 2 buveurs



III- Exemples

✓ **Base de Données vinicole :**

SCHEMA RELATIONNEL :

PRODUCTEUR (no-prod, nom-prod, ad-prod, region)

VIN (no-vin, nom-vin, cru, type, millesime, qte-stock, prix-unit, qte-prod, no-prod#)

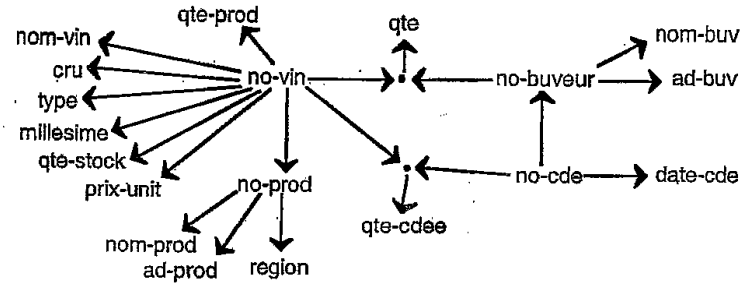
BUVEUR (no-buveur, nom-buv, ad-buv)

COMMANDE (no-cde, date-cde, no-buveur#)

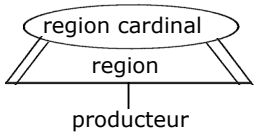
LIGNE-CDE (no-cde, no-vin, qte-cdee)

A-BU (no-buveur, no-vin, qte)

GRAPHE DE DEPENDANCES FONCTIONNELLES :



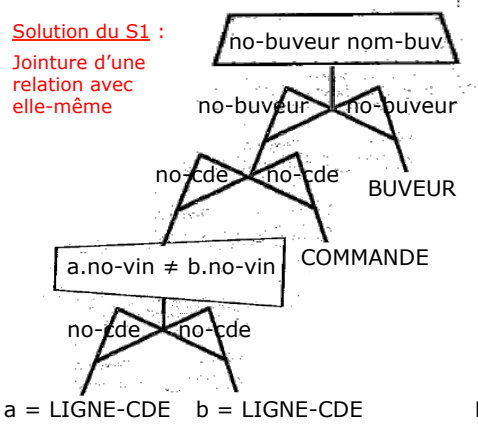
✓ **Requête 2-3** : nombre de producteurs par région



✓ **Requête 7 :** quels buveurs ont au moins une commande enregistrée qui concerne au moins 2 vins ?

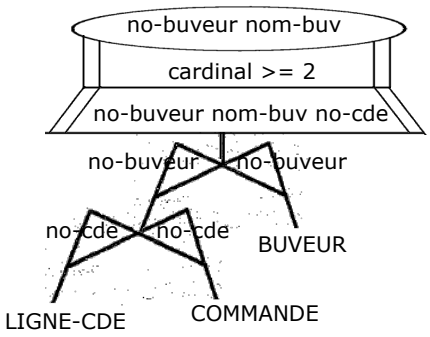
Solution du S1 :

Jointure d'une relation avec elle-même



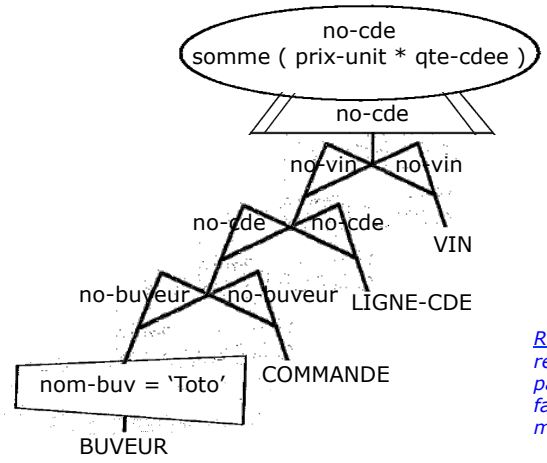
Solution avec partition :

Généralisable à plus de 2



- **Remarque** : le fait de mentionner *no-buveur* et *nom-buv* dans la partition est obligatoire pour les avoir dans le résultat ; sinon *no-cde* suffirait pour construire la partition.

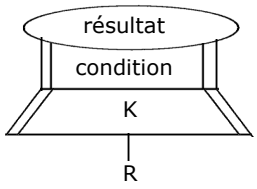
- **Requête 2-4** : montant de chacune des commandes passées par Toto



Remarque : La partie de la requête en dessous de la partition se construit de façon classique selon les méthodes vues au S1.

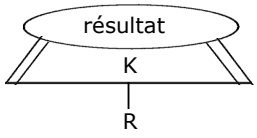
IV- Traduction en SQL

- **Schémas de traduction :**



⇒

```
SELECT résultat
FROM R
GROUP BY K
HAVING condition
```



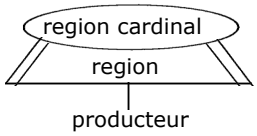
⇒

```
SELECT résultat
FROM R
GROUP BY K
```

- **Remarques** :

- Résultat = attributs de K et/ou opérateurs arithmétiques (+ alias éventuels) ;
- R peut résulter de différentes clauses SQL incluant notamment WHERE ;
- Il peut y avoir des variantes de syntaxe suivant les dialecte SQL (GROUP BY et HAVING pas toujours possibles dans une sous-interrogation).

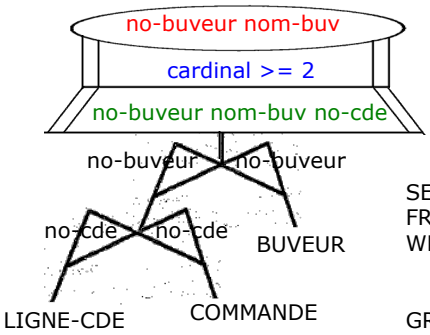
- **Exemples :**
- **Requête 2-3** :



⇒

```
SELECT region, COUNT(*)
FROM producteur
GROUP BY region
```

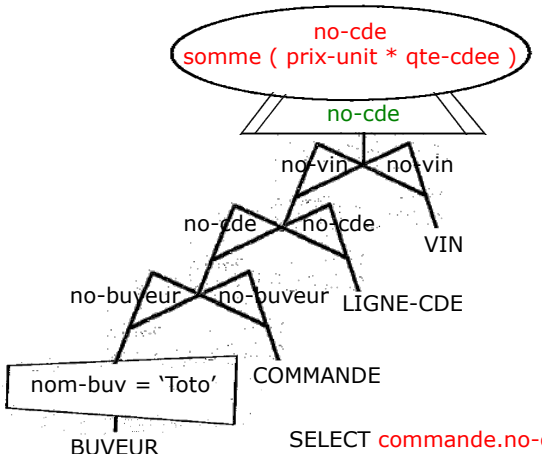
- **Requête 7 :**



⇒

```
SELECT DISTINCT buveur.no-buveur, nom-buv
FROM ligne-cde, commande, buveur
WHERE ligne-cde.no-cde = commande.no-cde
   AND commande.no-buveur =
                          buveur.no-buveur
GROUP BY buveur.no-buveur, nom-buv,
                          commande.no-cde
HAVING COUNT(*) >= 2
```

- **Requête 2-4** :



⇒

```
SELECT commande.no-cde, SUM ( prix-unit*qte-cdee )
FROM buveur, commande, ligne-cde, vin
WHERE nom-buv = 'Toto'
   AND buveur.no-buveur = commande.no-buveur
   AND ligne-cde.no-cde = commande.no-cde
   AND ligne-cde.no-vin = vin.no-vin
GROUP BY commande.no-cde
```

Chapitre 3 : L'UNION

I- Définition

- Soient R(X) et S(Y) 2 relations compatibles (cf définition de la différence) ; l'union de R et S, notée R **U** S, est formée de l'ensemble des n-uplets appartenant à R **ou** à S :

  R **U** S = { <x> / <x> ∈ R ou <x> ∈ S }

- **Notation graphique :**

- Exemple simple :

R

buveur	bière
Hugo	Guiness
Audrey	Grimbergen
Audrey	Guiness

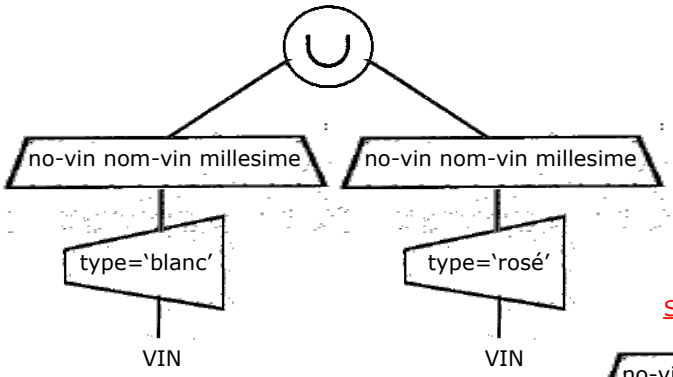
S

buveur	bière
Hugo	Grimbergen
Audrey	Grimbergen
Alexandre	Guiness

R **U** S

buveur	bière
Hugo	Guiness
Hugo	Grimbergen
Audrey	Grimbergen
Audrey	Guiness
Alexandre	Guiness

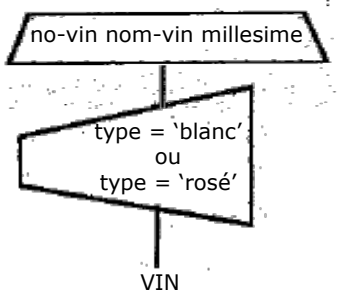
- **Requête 5** : liste des vins blancs et rosés



Solution équivalente à celle vue au S1 :

⇒ intérêt limité, pour l'instant...

Solution du S1 :



II- Traduction en SQL

- **Syntaxe :**

```
SELECT ...
FROM ...
...
  UNION
SELECT ...
FROM ...
...
  UNION
SELECT ...
FROM ...
...
```

Compatibilité nécessaire

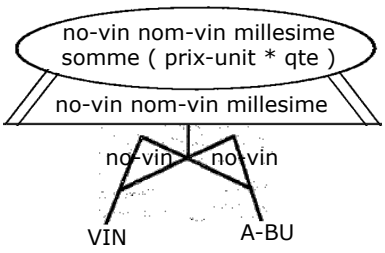
- **Remarques** :
  - suivant les dialectes SQL, le nombre d'UNION peut être limité ;
  - noms d'alias : ceux du 1er SELECT en général.

- **Exemple :** Requête 5 ci-dessus

```
SELECT no-vin, nom-vin, millesime
FROM vin
WHERE type = 'blanc'
  UNION
SELECT no-vin, nom-vin, millesime
FROM vin
WHERE type = 'rosé'
```

III- Cas 0

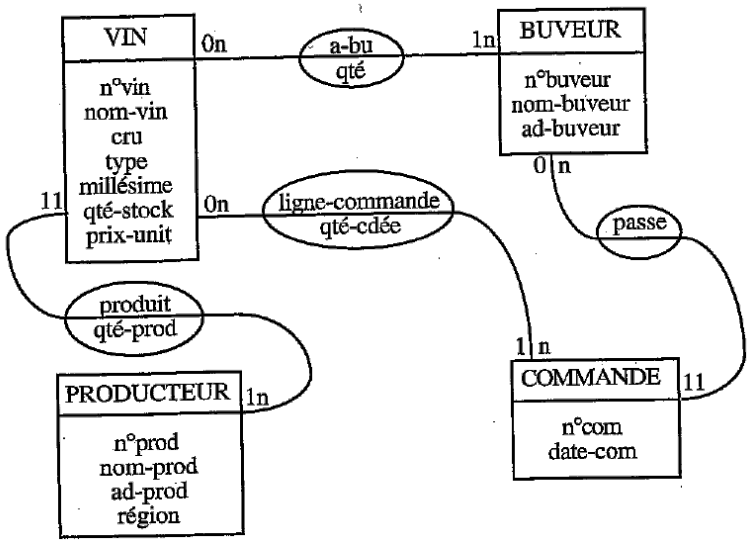
- Il existe des cas où l'UNION est indispensable pour traiter les différentes facettes d'un problème ; cela concerne en particulier le **Cas 0** que nous allons illustrer par un exemple.
- **Requête 2-5** : quel est le chiffre d'affaire associé à chaque vin?
- En première approche, nous allons utiliser les relations VIN et A-BU de façon à associer à chaque vin son chiffre d'affaire obtenu en cumulant, pour l'ensemble des buveurs ayant acheté ce vin, les produits qté*prix-unit. On obtient ainsi l'arbre suivant :



Cette solution est-elle la bonne?

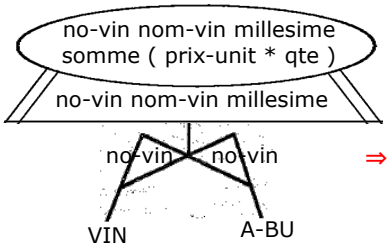
En apparence oui... Mais en fait, NON!!

Pourquoi? Pour l'expliquer, il nous faut revenir au Schéma Entité-Association de la BdD vinicole.



- La cardinalité minimum à 0 pour l'Association a-bu du côté de l'Entité VIN indique qu'il peut exister des vins pour lesquels il n'y a pas encore eu de vente. Dans ce cas, on a donc chiffre d'affaire nul. **Ce chiffre d'affaire nul doit-il se retrouver dans le résultat de la requête 2-5?**

- Ce chiffre d'affaire nul ne se retrouve pas dans la 1e version de la requête :



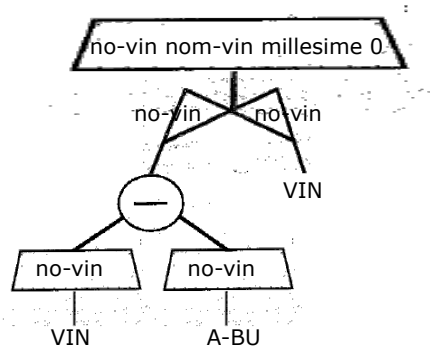
⇒

Cette requête donne en résultat le chiffre d'affaire des vins pour lesquels il y a eu des ventes.

=>Il nous faut donc déterminer les vins pour lesquels il n'y a pas eu de ventes et leur associer un chiffre d'affaire nul.

=> On obtient cette 2e partie du résultat par la requête suivante :

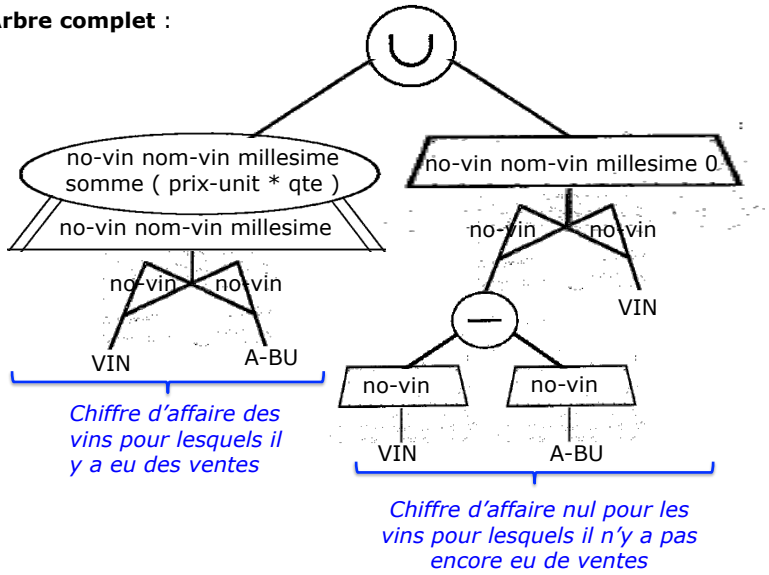
⇐



Le chiffre d'affaire égal à 0 est obtenu par la projection sur l'expression arithmétique élémentaire que représente la constante 0.

Les 2 résultats des 2 requêtes sont compatibles ; il suffit donc d'en faire l'UNION pour obtenir un résultat de requête complet.

✓ **Arbre complet** :

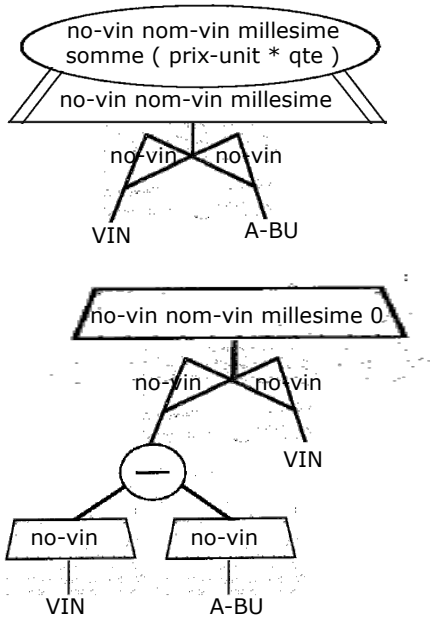


Chiffre d'affaire des vins pour lesquels il y a eu des ventes

Chiffre d'affaire nul pour les vins pour lesquels il n'y a pas encore eu de ventes

✓ **Remarque :** cette construction, qui constitue le **Cas 0**, est formulée de façon différente du cas général ; le concepteur de la requête doit fournir l'**effort d'analyse** nécessaire pour voir si le cas 0 doit être pris en compte.

✓ **Traduction en SQL** :




```
SELECT vin.no-vin, nom-vin, millesime,
       SUM ( prix-unit * qte )
FROM vin, a-bu
WHERE vin.no-vin = a-bu.no-vin
GROUP BY vin.no-vin, nom-vin, millesime
```

UNION

```
SELECT no-vin, nom-vin, millesime, 0
FROM vin
WHERE no-vin NOT IN
      ( SELECT no-vin
        FROM a-bu )
```

La constante 0 est interprétée comme étant une expression arithmétique.

Chapitre 4 : LA DIVISION

I- Définition

- ✓ Soient R(X,Y) et S(Y) 2 relations dans lesquelles X et Y représentent des ensembles non vides d'attributs ; la division de R par S, notée R **/** S, est une relation D(X) formée de tous les n-uplets <x> associés dans R à **tous** les n-uplets de S :
  R **/** S = { <x> / ∀ <y> ∈ S, alors <x,y> ∈ R }
- ✓ Exemple simple :

A :

buveur	bar
Hugo	Saint Jacques
Hugo	Irish Pub
Alexandre	La Promenade
Audrey	Irish Pub
Audrey	Saint Jacques
Audrey	La Promenade

B :

bar
Saint Jacques
Irish Pub
La Promenade

- Requête simple :
  Quels buveurs fréquentent tous les bars ?

⇒ A/B ⇒

buveur
Audrey

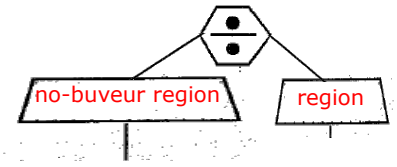
⇓

Littéralement : Quels buveurs sont associés dans A à tous les bars enregistrés dans B ?

- ✓ **Notation graphique :**

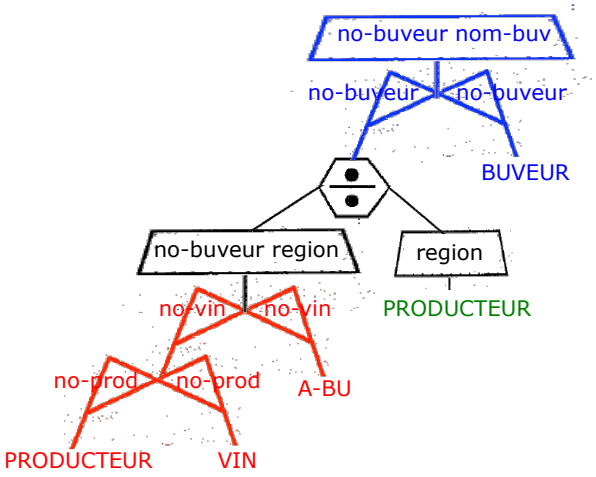
II- Exemple

- ✓ **Requête 2-6** : quels buveurs ont déjà acheté des vins provenant de toutes les régions en enregistrées ?
- ✓ Démarche : on pose d'abord la division ; il s'agit de diviser des couples (no-buveur, region) par region :



- ✓ Puis on complète en bas à gauche , en bas à droite, au dessus :

- **Arbre complété** :



III- Traduction en SQL

- La traduction de la Division en SQL va nécessiter une réécriture des requêtes.
- Par ailleurs, elle s'appuie sur le cas 3 des sous-conditions possibles dans une clause WHERE (cf S1 – Chapitre 3 – Paragraphe IX). Rappel :

- **3-** sous-conditions avec sous-interrogation
  - 3-1- vérifier l'appartenance (ou la non appartenance) de la valeur d'un attribut au résultat d'une sous-interrogation SELECT
    => IN et NOT IN
  - 3-2- vérifier l'existence (ou non) du résultat d'une sous-interrogation SELECT
- **Syntaxe du cas 3-2 :**

⇒ EXISTS / NOT EXISTS (SELECT ...)

- **Reprise de la requête 2-6** : on commence par la réécrire avec une double négation :

Quels sont les buveurs pour lesquels il n'existe pas de région dans laquelle ils n'auraient acheté aucun vin ?

✓ **Ensuite, début de la démarche de traduction en SQL** :

- on traite buveur par buveur ;
- puis, on détermine l'ensemble des régions d'où proviennent les vins qu'a déjà acheté le buveur en cours de traitement :

```
SELECT no-buveur, nom-buv
FROM buveur
```

⇒ *Ensemble des régions d'où proviennent les vins qu'a déjà acheté le buveur en cours de traitement*

```
( SELECT region
  FROM producteur, vin, a-bu
  WHERE producteur.no-prod = vin.no-prod
     AND vin.no-vin = a-bu.no-vin
     AND a-bu.no-buveur = buveur.no-buveur )
```

⇑ *Référence au buveur en cours de traitement*

✓ **Suite de la démarche** :

- ensuite, on détermine par différence l'ensemble des régions dans lesquelles le buveur en cours de traitement n'a acheté aucun vin :

```
SELECT no-buveur, nom-buv
FROM buveur
```

⇒ *Ensemble des régions dans lesquelles le buveur en cours de traitement n'a acheté aucun vin*

```
( SELECT region
  FROM producteur
  WHERE region NOT IN
        ( SELECT region
          FROM producteur, vin, a-bu
          WHERE producteur.no-prod = vin.no-prod
             AND vin.no-vin = a-bu.no-vin
             AND a-bu.no-buveur = buveur.no-buveur ) )
```

✓ **Fin de la démarche** :

- on a déterminé l'ensemble des régions dans lesquelles le buveur en cours de traitement n'a acheté aucun vin ;

=> on veut que cet ensemble soit vide pour retenir le buveur en cours de traitement :

```
SELECT no-buveur, nom-buv
FROM buveur
WHERE NOT EXISTS
    ( SELECT region
      FROM producteur
      WHERE region NOT IN
          ( SELECT region
            FROM producteur, vin, a-bu
            WHERE producteur.no-prod = vin.no-prod
              AND vin.no-vin = a-bu.no-vin
              AND a-bu.no-buveur = buveur.no-buveur ) )
```

✓ **Récapitulatif** :

```
SELECT no-buveur, nom-buv
FROM buveur
WHERE NOT EXISTS                                   ⇐
    ( SELECT region
      FROM producteur
      WHERE region NOT IN
          ( SELECT region
            FROM producteur, vin, a-bu
            WHERE producteur.no-prod = vin.no-prod
              AND vin.no-vin = a-bu.no-vin
              AND a-bu.no-buveur = buveur.no-buveur ) )
```

⇒

2 *- Ensemble des régions dans lesquelles le buveur en cours de traitement n'a acheté aucun vin*

3 - on veut que cet ensemble soit vide pour retenir le buveur en cours de traitement :

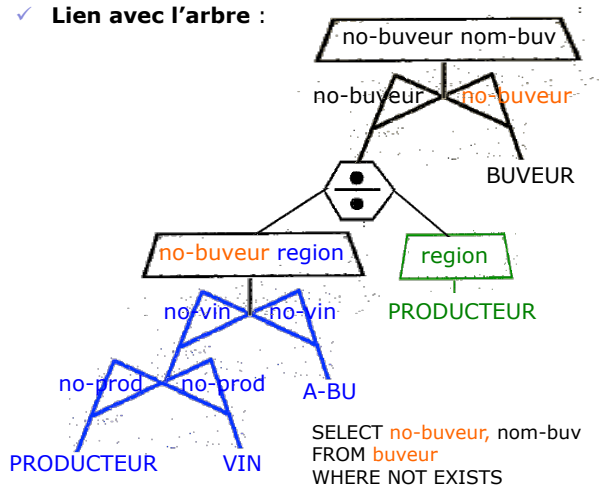
⇒

1 *- Ensemble des régions d'où proviennent les vins qu'a déjà acheté le buveur en cours de traitement*

⇑

Référence au buveur en cours de traitement

✓ **Lien avec l'arbre** :




```
SELECT no-buveur, nom-buv
FROM buveur
WHERE NOT EXISTS
     ( SELECT region
       FROM producteur
       WHERE region NOT IN
             ( SELECT region
               FROM producteur, vin, a-bu
               WHERE producteur.no-prod = vin.no-prod
                  AND vin.no-vin = a-bu.no-vin
                  AND a-bu.no-buveur = buveur.no-buveur ) )
```

✓ **Simulation du fonctionnement de la requête** :

✓ Soit un état de la BdD avec 4 buveurs b1, b2, b3, b4 associés chacun, en passant par *producteur*, *vin*, et *a-bu,* aux régions suivantes :
b1 -> { Alsace, Champagne }
b2 -> { Alsace }
b3 -> { Alsace, Moselle, Champagne, Bourgogne }
b4 -> { Alsace, Moselle, Bourgogne }

✓ **Ensemble des régions =>**

region
Alsace
Moselle
Champagne
Bourgogne

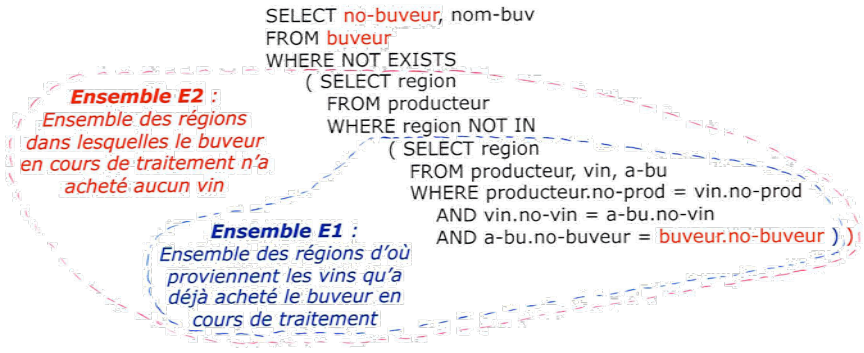
```
SELECT no-buveur, nom-buv
FROM buveur
WHERE NOT EXISTS
     ( SELECT region
       FROM producteur
       WHERE region NOT IN
             ( SELECT region
               FROM producteur, vin, a-bu
               WHERE producteur.no-prod = vin.no-prod
                  AND vin.no-vin = a-bu.no-vin
                  AND a-bu.no-buveur = buveur.no-buveur ) )
```

Ensemble E2 : *Ensemble des régions dans lesquelles le buveur en cours de traitement n'a acheté aucun vin*

Ensemble E1 : *Ensemble des régions d'où proviennent les vins qu'a déjà acheté le buveur en cours de traitement*

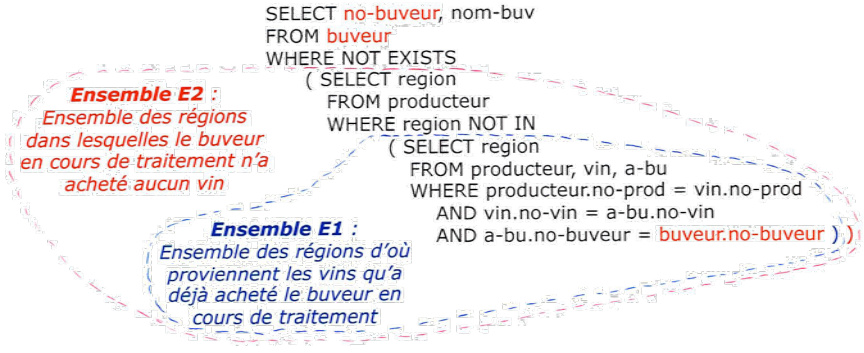
- **Etapes du fonctionnement de la requête :**
- Le système traite chaque buveur l'un après l'autre.
- Pour le buveur b1 :
  E1 = { Alsace, Champagne }
  => E2 = { Moselle, Bourgogne }
  E2 ≠ Ø => NOT EXISTS -> FALSE => le buveur b1 n'est pas sélectionné
- Le système passe au buveur b2 :
  E1 = { Alsace }
  => E2 = { Moselle, Champagne, Bourgogne }
  E2 ≠ Ø => NOT EXISTS -> FALSE => le buveur b2 n'est pas sélectionné

region
Alsace
Moselle
Champagne
Bourgogne



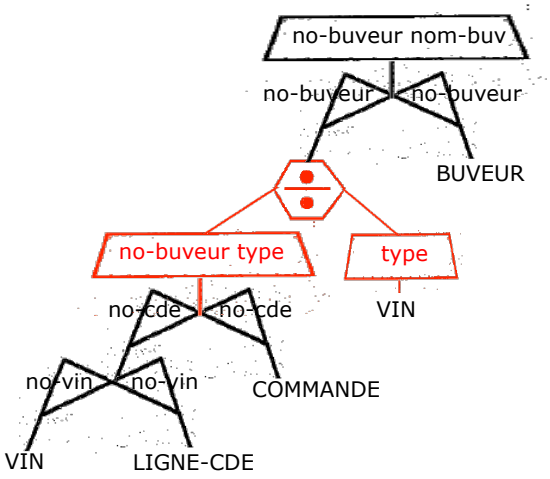
- **Etapes du fonctionnement de la requête - suite :**
- Pour le buveur b3 :
  E1 = { Alsace, Moselle, Champagne, Bourgogne }
  => E2 = Ø
  E2 = Ø => NOT EXISTS -> TRUE => le buveur b3 est sélectionné
- Le système passe au buveur b4 :
  E1 = { Alsace, Moselle, Bourgogne }
  => E2 = { Champagne }
  E2 ≠ Ø => NOT EXISTS -> FALSE => le buveur b4 n'est pas sélectionné

region
Alsace
Moselle
Champagne
Bourgogne



IV- Exemple 2

- **Requête 2-7** : quels buveurs ont des commandes en cours concernant tous les types de vin ?
- On pose d'abord la division ; il s'agit de diviser des couples (no-buveur, type) par type ; puis on complète en bas à gauche, en bas à droite, et au dessus :



- **Traduction en SQL** :

Reformulation : Quels sont les buveurs pour lesquels il n'existe pas de type de vin qui ne se retrouveraitt pas dans au moins une des commandes en cours de ces buveurs ?

```
SELECT no-buveur, nom-buv
FROM buveur
WHERE NOT EXISTS                    ⇐
    ( SELECT type
      FROM vin
      WHERE type NOT IN
          ( SELECT type
            FROM vin, ligne-cde, commande
            WHERE vin.no-vin = ligne-cde.no-vin
              AND ligne-cde.nocde = commande.no-cde
              AND commande.no-cde = buveur.no-buveur ) )
```

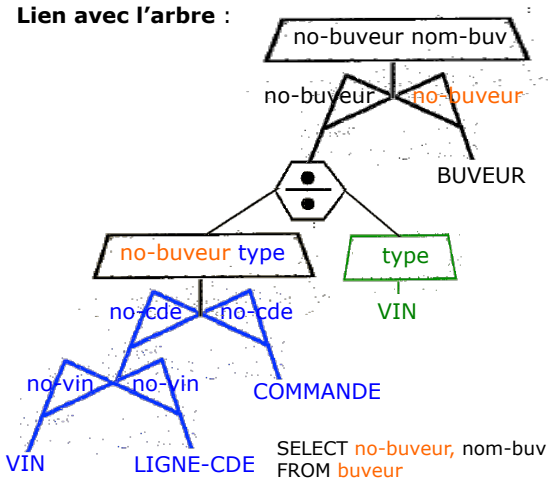
⇒ ***2*** *- Ensemble des types de vin pour lesquels le buveur en cours de traitement n'a pas de commande en cours*

⇒ ***1*** *- Ensemble des types de vin pour lesquels le buveur en cours de traitement a des commandes en cours*

3 - on veut que cet ensemble soit vide pour retenir le buveur en cours de traitement :

⇑ *Référence au buveur en cours de traitement*

- **Lien avec l'arbre** :



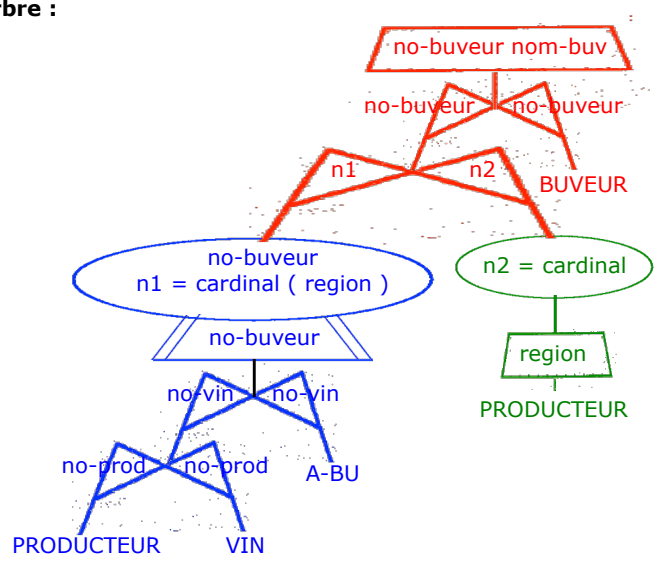

```
SELECT no-buveur, nom-buv
FROM buveur
WHERE NOT EXISTS
    ( SELECT type
      FROM vin
      WHERE type NOT IN
            ( SELECT type
              FROM vin, ligne-cde, commande
              WHERE vin.no-vin = ligne-cde.no-vin
                 AND ligne-cde.nocde = commande.no-cde
                 AND commande.no-cde = buveur.no-buveur ) )
```

V- Approche arithmétique

- L'approche ensembliste s'appuyant sur EXISTS vue ci-dessus peut être remplacée par une approche arithmétique ;
- A(X,Y) / B(Y) => on sélectionne les n-uplets <x> associé dans A à autant de n-uplets <y> qu'il en existe dans B (à condition que ce soit les mêmes valeurs de <y>).
- **Exemple 1** : requête 2-6

Reformulation : Quels sont les buveurs qui ont acheté des vins provenant d'autant de régions différentes qu'il en existe ?

✓ **Arbre :**



✓ **Traduction en SQL (2 solutions)** :

```
CREATE TEMPORARY TABLE reltemp1
SELECT no-buveur, COUNT ( DISTINCT
region ) n1
FROM producteur, vin, a-bu
WHERE producteur.no-prod = vin.no-prod
    AND vin.no-vin =a-bu.no-vin
    GROUP BY no-buveur
;
CREATE TEMPORARY TABLE reltemp2
SELECT COUNT ( DISTINCT region ) n2
FROM producteur
;
SELECT buveur.no-buveur, nom-buv
FROM buveur, reltemp1, reltemp2
WHERE buveur.no-buveur = a-bu.no-buveur
    AND n1 = n2
;
DROP TABLE reltemp1
;
DROP TABLE reltemp2
```

OU

```
SELECT buveur.no-buveur, nom-buv
FROM buveur, reltemp1
WHERE buveur.no-buveur
                = a-bu.no-buveur
    AND n1 IN
      ( SELECT COUNT ( DISTINCT
                        region )
        FROM vin )
;
DROP TABLE reltemp1
```

Retour sur l'exemple de simulation du fonctionnement de la requête :

reltemp1

no-buveur	n1
b1	2
b2	1
b3	4
b4	3

*

⇓

b3

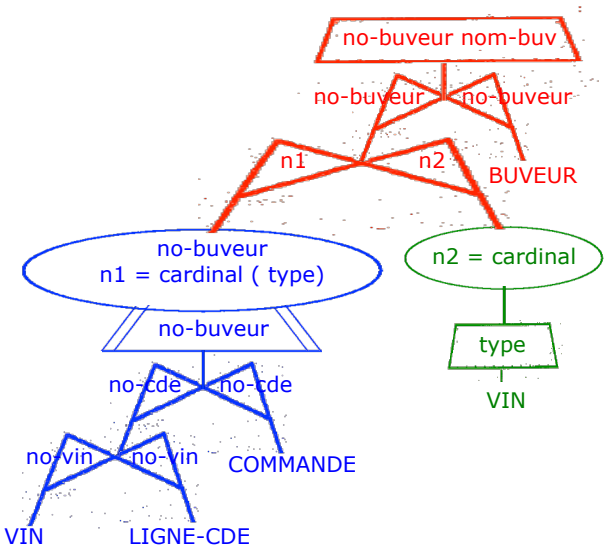
reltemp2

n1
4

- **Exemple 2** : requête 2-7

  Reformulation : Quels sont les buveurs qui ont des commandes en cours concernant des vins d'autant de types différents qu'il en existe ?

- **Arbre :**



- **Traduction en SQL (2 solutions)** :

```
CREATE TEMPORARY TABLE reltemp1
SELECT no-buveur, COUNT ( DISTINCT
type ) n1
FROM vin, ligne-cde, commande
WHERE vin.no-vin = ligne-cde.no-vin
    AND ligne-cde.no-commande
            = commande.no-commande
    GROUP BY no-buveur
;
CREATE TEMPORARY TABLE reltemp2
SELECT COUNT ( DISTINCT type ) n2
FROM vin
;
SELECT buveur.no-buveur, nom-buv
FROM buveur, reltemp1, reltemp2
WHERE buveur.no-buveur = a-bu.no-buveur
    AND n1 = n2
;
DROP TABLE reltemp1
;
DROP TABLE reltemp2
```

OU

```
SELECT buveur.no-buveur, nom-buv
FROM buveur, reltemp1
WHERE buveur.no-buveur
                = a-bu.no-buveur
    AND n1 IN
      ( SELECT COUNT ( DISTINCT
                        type )
        FROM vin )
;
DROP TABLE reltemp1
```

Chapitre 5 : EXEMPLES

I- Optimisation

- Il s'agit d'aborder l'optimisation des requêtes en permettant la réutilisation de résultats intermédiaires, ce qui peut conduire à des structures de requête qui ne sont plus des arbres.

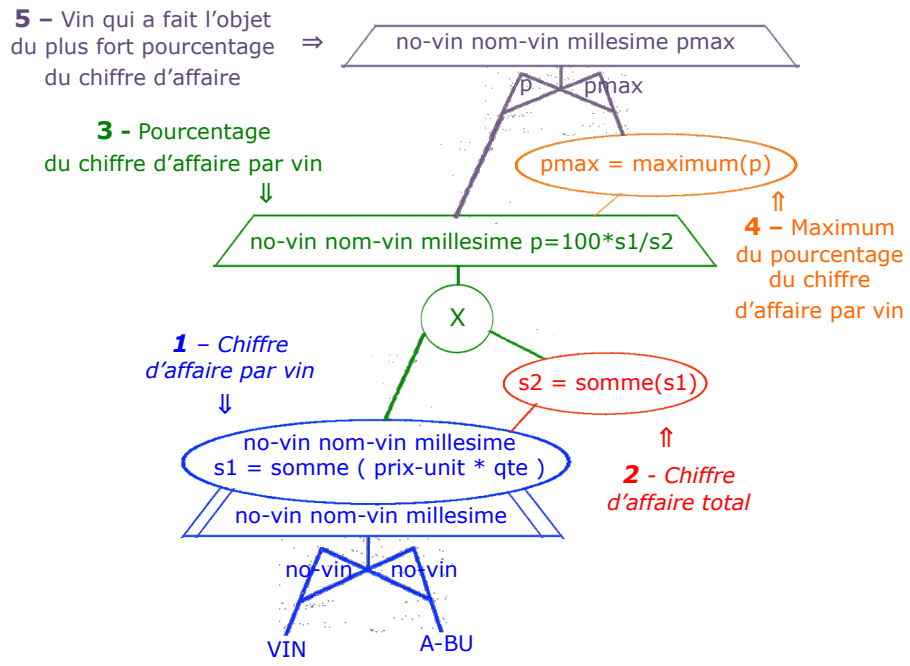
  Cette optimisation s'appuie notamment sur le fait que le résultat de chaque opération de l'algèbre relationnelle constitue une relation qui peut faire l'objet d'autant (donc plusieurs le cas échéant) d'utilisations que nécessaire.

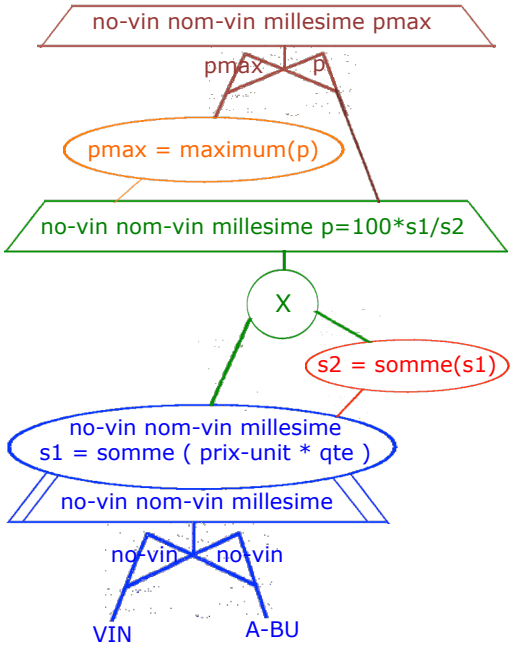
- **Requête 2-8** : quel vin a fait l'objet du plus fort pourcentage du chiffre d'affaire total ?

- Cette requête nécessite de s'appuyer sur :
  - le chiffre d'affaire par vin,
  - le chiffre d'affaire total,
  - le pourcentage,
  - le maximum du pourcentage.

- Il s'agit donc de construire une requête dans laquelle on minimise la répétition de recherches d'information et de calculs de même nature ; dans le cas de la requête 2-8, cet objectif concerne en particulier le calcul du chiffre d'affaire par vin.

- **Arbre** :

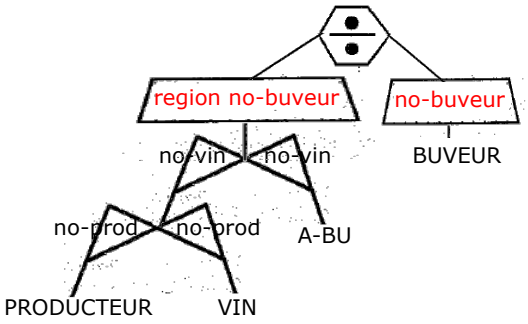





```
CREATE TEMPORARY TABLE reltemp1
SELECT no-vin, nom-vin, millesime,
            SUM ( prix-unit * qte ) s1
FROM vin.vin, a-bu
WHERE vin.no-vin = a-bu.no-vin
GROUP BY vin.no-vin, nom-vin, millesime
;
CREATE TEMPORARY TABLE reltemp2
SELECT SUM ( s1 ) s2
FROM reltemp1
;
CREATE TEMPORARY TABLE reltemp3
SELECT no-vin, nom-vin, millesime,
                  100*s1/s2 p
FROM reltemp1, reltemp2
;
SELECT no-vin, nom-vin, millesime, p
FROM reltemp3
WHERE p IN ( SELECT MAX (p)
                  FROM reltemp3 )
;
DROP TABLE reltemp1
;
DROP TABLE reltemp2
;
DROP TABLE reltemp3
```

II- Division sans partie supérieure

- **Requête 2-9** : quelles sont les régions pour lesquelles tous les buveurs enregistrés ont acheté des vins qui en sont issus ?
- On pose d'abord la division ; il s'agit de diviser des couples (region, no-buveur) par no-buveur ; puis on complète là où c'est nécessaire :



- Le résultat est un ensemble de régions issues de la relation PRODUCTEUR en bas à gauche ; comment le transposer en SQL ?

✓ **Traduction en SQL** :

Reformulation : Quelles sont les régions pour lesquelles il n'existe pas de buveur qui n'aurait pas acheté de vin provenant de cette région ?

✓ **1° approche** : utilisation de 2 alias de la relation PRODUCTEUR

2 - Ensemble des buveurs qui n'ont pas acheté de vins provenant de la région en cours de traitement

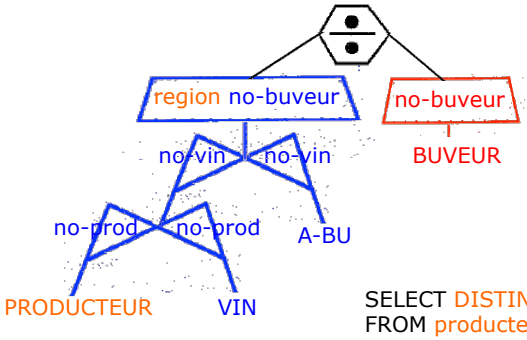
```
SELECT DISTINCT region
FROM producteur p1
WHERE NOT EXISTS                    ⇐
    ( SELECT no-buveur
      FROM buveur
      WHERE no-buveur NOT IN
            ( SELECT no-buveur
              FROM a-bu, vin, producteur p2
              WHERE a-bu.no-vin = vin.no-vin
                AND vin.no-prod = p2.no-prod
                AND p2.region = p1.region ) )
```

⇒ (2) ⇒ (1)

1 - Ensemble des buveurs qui ont acheté des vins provenant de la région en cours de traitement

3 - on veut que cet ensemble soit vide pour retenir la région en cours de traitement :

✓ **Lien avec l'arbre** :




```
SELECT DISTINCT region
FROM producteur p1
WHERE NOT EXISTS
    ( SELECT no-buveur
      FROM buveur
      WHERE no-buveur NOT IN
            ( SELECT no-buveur
              FROM a-bu, vin, producteur p2
              WHERE a-bu.no-vin = vin.no-vin
                AND vin.no-prod = p2.no-prod
                AND p2.region = p1.region ) )
```

- Cette première approche peut se révéler peu efficace ; en effet, pour chaque producteur (et non pour chaque région), les sous-interrogations sont exéxutées spécifiquement, et donc répétées pour chaque producteur issu de la même région ; pour éviter cela, une 2e approche est possible :

- **2° approche** : utilisation d'une relation temporaire pour isoler les régions

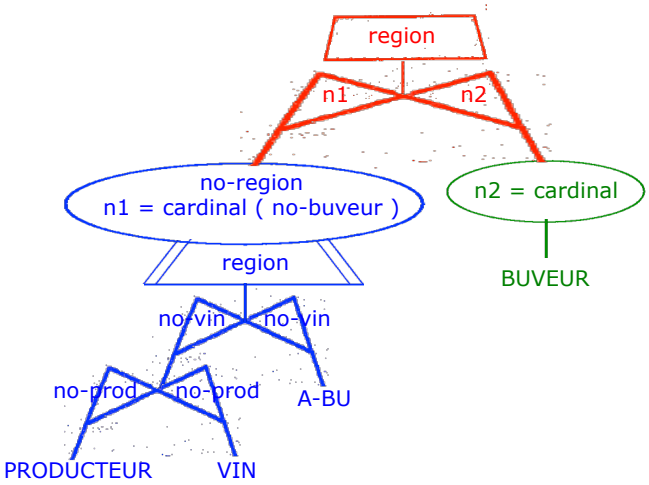
2 - Ensemble des buveurs qui n'ont pas acheté de vins provenant de la région en cours de traitement

1 - Ensemble des buveurs qui ont acheté des vins provenant de la région en cours de traitement

```
CREATE TEMPORARY TABLE reltemp
SELECT DISTINCT region
FROM producteur
;
SELECT region
FROM reltemp
WHERE NOT EXISTS                    <=  3 - on veut que cet ensemble soit vide pour retenir la région en cours de traitement :
=>  ( SELECT no-buveur
      FROM buveur
      WHERE no-buveur NOT IN
            ( SELECT no-buveur
              FROM a-bu, vin, producteur
              WHERE a-bu.no-vin = vin.no-vin
                 AND vin.no-prod = p2.no-prod
                 AND producteur.region = reltemp.region ) )
;
DROP TABLE reltemp
```

- **3° approche** : utilisation de l'approche arithmétique

Reformulation : Quelles sont les régions pour lesquelles il y a autant de buveurs-acheteurs différents qu'il en existe ?



✓ **Traduction en SQL (2 solutions)** :

```
CREATE TEMPORARY TABLE reltemp1
SELECT region, COUNT ( DISTINCT no-buveur ) n1
FROM producteur, vin, a-bu
WHERE producteur.no-prod = vin.no-prod
    AND vin.no-vin =a-bu.no-vin
GROUP BY region
;
CREATE TEMPORARY TABLE reltemp2
SELECT COUNT (*) n2
FROM buveur
;
SELECT region
FROM reltemp1, reltemp2
WHERE n1 = n2
;
DROP TABLE reltemp1
;
DROP TABLE reltemp2
```

OU

```
SELECT region
FROM reltemp1
WHERE n1 IN
      ( SELECT COUNT ( DISTINCT
                              region )
        FROM producteur )
;
DROP TABLE reltemp1
```

III- Etablir un classement

1- Problématique

✓ **Objectif** : ordonner les résultats d'une requête (utilisation de ORDER BY) et afficher automatiquement le classement (1er, 2e, ...)

✓ **Requête 2-10** : quelles sont les 10 vins qui représentent les pourcentages du chiffre d'affaire total les plus élevés et dans quel ordre ?

✓ **Démarche** : elle peut être décomposée en 3 facettes

1- sélectionner les données à ordonner (pour la requête ci-dessus, établir le chiffre d'affaire par vin) ; pour ce faire, on utilise les démarches habituelles ;

2- ordonner ces données ;

3- départager les ex æquo.

✓ Avant de revenir sur la requête 2-10, nous allons commencer par étudier les points 2 et 3.

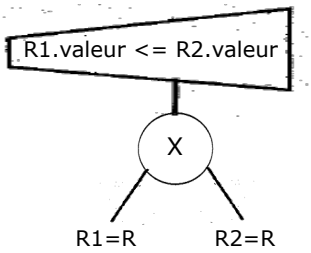
2- Ordonner un ensemble de données

- Nous allons exposer une démarche possible à partir d'un ensemble de données simples. Soit une relation R (ident, valeur), avec le contenu suivant qu'il va s'agir d'ordonner (ordre décroissant) et de classer :

R :

ident	valeur
i1	50
i2	130
i3	15

- Nous allons commencer en utilisant deux alias R1 et R2 de la relation R à l'aide desquels nous allons, à partir du produit cartésien R1 x R2, associer à chaque n-uplet de R1 tous les n-uplets de R2 qui ont une valeur égale ou supérieure :



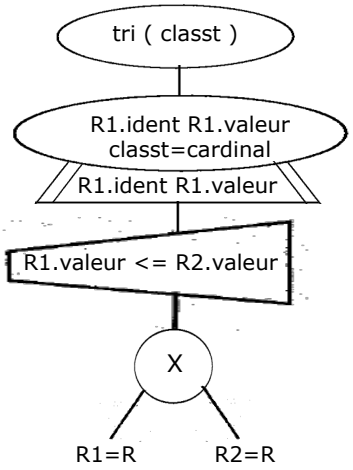
- On obtient ainsi le résultat suivant :

R1 : ident	R1 : valeur	R2 : ident	R2 : valeur
i1	50	i1	50
i1	50	i2	130
~~i1~~	~~50~~	~~i3~~	~~15~~
~~i2~~	~~130~~	~~i1~~	~~50~~
i2	130	i2	130
~~i2~~	~~130~~	~~i3~~	~~15~~
i3	15	i1	50
i3	15	i2	130
i3	15	i3	15

- Avec la relation ainsi obtenue, on opère une partition suivant les valeurs de R1.ident ; le nombre de n-uplets de chaque sous-relation ainsi obtenue correspond à son classement :

R1.ident	R1.valeur	R2.ident	R2.valeur
i1	50	i1	50
i1	50	i2	130
i2	130	i2	130
i3	15	i1	50
i3	15	i2	130
i3	15	i3	15

✓ **Arbre et traduction en SQL** :




```
SELECT COUNT(*) classt, r1.ident, r1.valeur
FROM r r1, r r2
WHERE r1.valeur <= r2.valeur
GROUP BY r1.ident, r1.valeur
ORDER BY classt
```

⇓

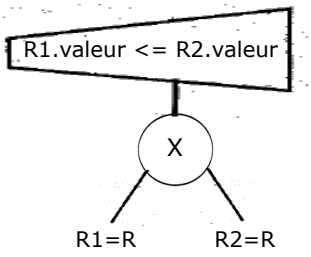
classt	ident	valeur
1	i2	130
2	i1	50
3	i3	15

3- Prendre en compte les ex æquo éventuels

✓ Reprenons la démarche à partir d'un ensemble de données avec ex æquo et correspondant à la même relation R (ident, valeur) :

ident	valeur
i1	50
i2	130
i3	15
i4	15
i5	130

✓ Nous allons reprendre la même démarche avec deux alias R1 et R2 de la relation R à l'aide desquels nous allons, à partir du produit cartésien R1 x R2, associer à chaque n-uplet de R1 tous les n-uplets de R2 qui ont une valeur égale ou supérieure, puis adapter le traitement du tableau obtenu :



✓ On obtient ainsi le résultat suivant :

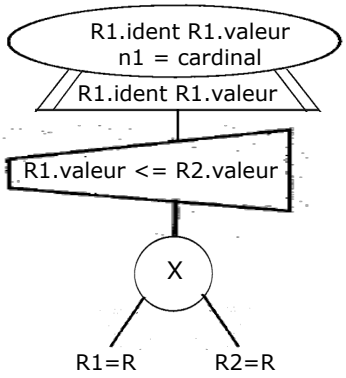
R1.ident	R1.valeur	R2.ident	R2.valeur
i1	50	i1	50
i1	50	i2	130
~~i1~~	~~50~~	~~i3~~	~~15~~
~~i1~~	~~50~~	~~i4~~	~~15~~
i1	50	i5	130
~~i2~~	~~130~~	~~i1~~	~~50~~
i2	130	i2	130
~~i2~~	~~130~~	~~i3~~	~~15~~
~~i2~~	~~130~~	~~i4~~	~~15~~
i2	130	i5	130
i3	15	i1	50
i3	15	i2	130
i3	15	i3	15
i3	15	i4	15
i3	15	i5	130
i4	15	i1	50
i4	15	i2	130
i4	15	i3	15
i4	15	i4	15
i4	15	i5	130
~~i5~~	~~130~~	~~i1~~	~~50~~
i5	130	i2	130
~~i5~~	~~130~~	~~i3~~	~~15~~
~~i5~~	~~130~~	~~i4~~	~~15~~
i5	130	i5	130

✓ Comme précédemment, on opère une partition suivant les valeurs de R1.ident :

R1.ident	R1.valeur	R2.ident	R2.valeur
i1	50	i1	50
i1	50	i2	130
i1	50	i5	130
i2	130	i2	130
i2	130	i5	130
i3	15	i1	50
i3	15	i2	130
i3	15	i3	15
i3	15	i4	15
i3	15	i5	130
i4	15	i1	50
i4	15	i2	130
i4	15	i3	15
i4	15	i4	15
i4	15	i5	130
i5	130	i2	130
i5	130	i5	130

✓ Le nombre d'éléments correspond au classement s'il n'y a pas d'ex æquo ; dans les autres cas, il faut prendre en compte le nombre d'ex æquo et opérer un retraitement :

✓ **Arbre** :



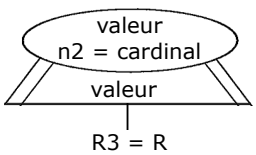
⇒

n1	R1.ident	R1.valeur
3	i1	50
2	i2	130
5	i3	15
5	i4	15
2	i5	130

⇓

✓ Puis on va calculer pour chaque valeur son nombre d'occurrences, puis associer ce nombre à chaque ligne du tableau ci-dessus :

⇓

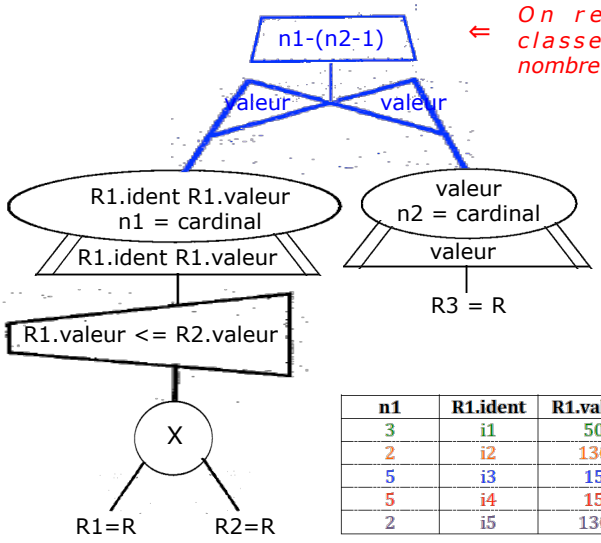


⇐

valeur	n2
50	1
130	2
15	2

n2 = nombre d'ex aequo

✓ **Prise en compte du nombre d'ex aequo** :

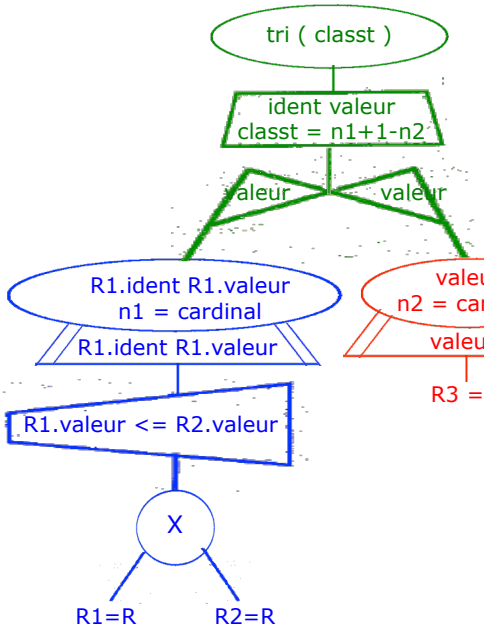


⇐ *On retire de n1 (le classement initial) le nombre d'ex æquo - 1*

n1	R1.ident	R1.valeur	n2	n1+1-n2
3	i1	50	1	3
2	i2	130	2	1
5	i3	15	2	4
5	i4	15	2	4
2	i5	130	2	1

⇑ *classt*

✓ **Arbre et traduction en SQL** :



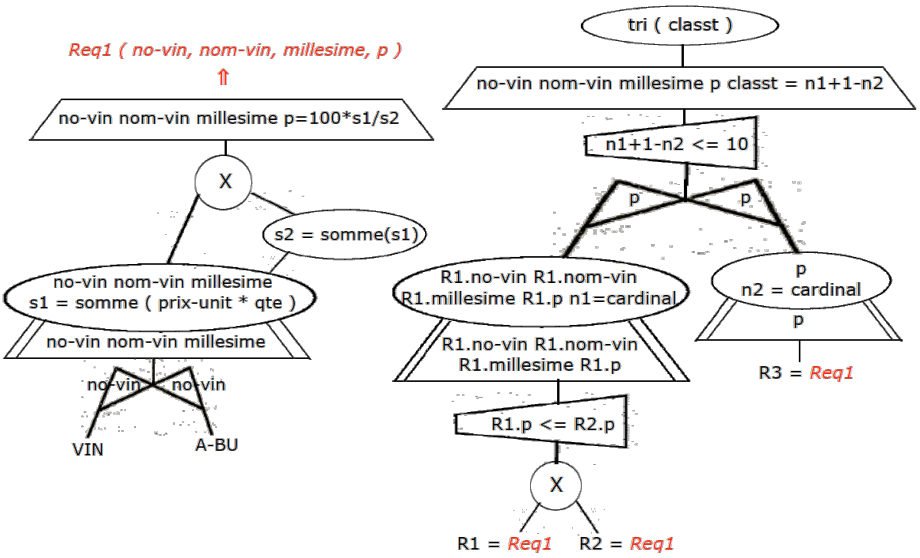

```
CREATE TEMPORARY TABLE rt1
SELECT COUNT(*) n1, r1.ident, r1.valeur
FROM r r1, r r2
WHERE r1.valeur <= r2.valeur
GROUP BY r1.ident, r1.valeur
ORDER BY classt
;
CREATE TEMPORARY TABLE rt2
SELECT valeur, count(*) n2
FROM r
GROUP BY valeur
;
```

classt	R1.ident	R1.valeur
1	i2	130
1	i5	130
3	i1	50
4	i3	15
4	i4	15

```
SELECT n1+1-n2 classt, ident, rt1.valeur
FROM rt1, rt2
WHERE rt1.valeur = rt2.valeur
ORDER BY classt
;
DROP TABLE rt1
;
DROP TABLE rt2
```

4- Retour sur la requête 2-10

✓ **Requête 2-10** : quelles sont les 10 vins qui représentent les pourcentages du chiffre d'affaire total les plus élevés et dans quel ordre ?



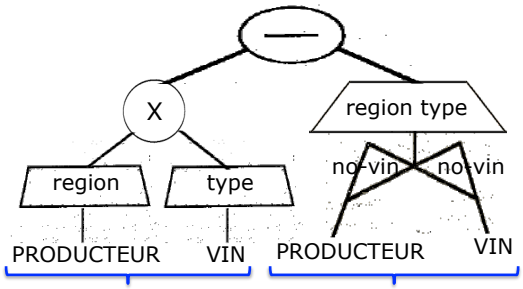
✓ **Traduction en SQL** :

```
CREATE TEMPORARY TABLE reltemp1
SELECT no-vin, nom-vin, millesime,
          SUM ( prix-unit * qte ) s1
FROM vin.vin, a-bu
WHERE vin.no-vin = a-bu.no-vin
GROUP BY vin.no-vin, nom-vin,
millesime
;
CREATE TEMPORARY TABLE reltemp2
SELECT SUM ( s1 ) s2
FROM reltemp1
;
CREATE TEMPORARY TABLE Req1
SELECT no-vin, nom-vin, millesime,
                100*s1/s2 p
FROM reltemp1, reltemp2
;
DROP TABLE reltemp1
;
DROP TABLE reltemp2
```

```
CREATE TEMPORARY TABLE rt1
SELECT r1.no-vin, r1.nom-vin, r1.millesime,
        r1.p, COUNT(*) n1
FROM Req1 r1, Req1 r2
WHERE r1.p <= r2.p
GROUP BY r1.no-vin, r1.nom-vin,
          r1.millesime, r1.q
;
CREATE TEMPORARY TABLE rt2
SELECT p, count(*) n2
FROM Req1
GROUP BY p
;
SELECT n1+1-n2 classt, rt1.no-vin,
        rt1.nom-vin, rt1.millesime, rt1.p
FROM rt1, rt2
WHERE rt1.p = rt2.p
   AND n1+1-n2 <=10
ORDER BY classt
;
DROP TABLE rt1
;
DROP TABLE rt2
;
DROP TABLE Req1
```

IV- Différence portant sur plusieurs attributs

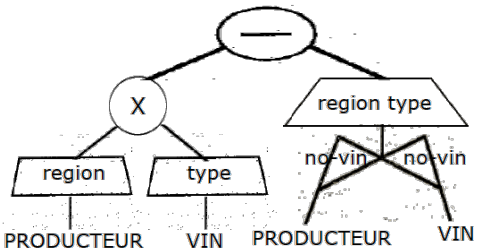
- **Requête 2-11** : pour chaque région, quels sont lesquels les types de vin pour lesquels aucun vin n'est proposé à la vente ?
- Il s'agit donc d'opérer une différence entre tous les couples (region type) possibles et l'ensemble des couples (region type) existant dans l'ensemble des vins associés à leur producteur.



Ensemble des couples (region type) possibles — *Ensemble des couples (region type) existants*

- **Traduction en SQL** : utilisation de NOT EXISTS
- L'utilisation du NOT IN, qui porte sur un seul attribut, n'est pas possible dans ce contexte ; on va donc utiliser NOT EXISTS en s'appuyant sur une reformulation de la requête :

Quels sont les couples (region type) pour lesquels il n'existe pas de valeurs correspondantes dans l'ensemble des vins associés à leur producteur ?



1 *- Ensemble des vins associés à leur producteur correspondant au couple (region type) en cours de traitement*

```
CREATE TEMPORARY TABLE reltemp1
SELECT DISTINCT region
FROM producteur
;
CREATE TEMPORARY TABLE reltemp2
SELECT DISTINCT type
FROM vin
;
SELECT region, type
FROM reltemp1, reltemp2
WHERE NOT EXISTS
      ( SELECT *
        FROM producteur, vin
        WHERE producteur.no-vin
              = vin.no-vin
          AND producteur.region
              = reltemp1.region
          AND vin.type = reltemp2.type )
;
DROP TABLE reltemp1
;
DROP TABLE reltemp2
```

⇐ ***2*** *– Le couple (region type) en cours de traitement est retenu si l'ensemble* ***1*** *est vide*

⇒

V- Synthèse : décomposition de requête

- ✓ **Requête 2-12** : pour chaque région, quel est le nombre de vins associés à chaque type et le chiffre d'affaire correspondant ?
- ✓ Il s'agit de construire des quadruplets (region, type, nb-vin, chiffre-affaire) pour tous les couples (region, type) pour lesquels des vins existent.
- ✓ Cette requête nécessite une analyse qui la décompose en plusieurs sous-cas :

  1- déterminer le nombre de vins différents par couple (region type) ; pour cette partie, le cas 0 n'est pas pris en compte ;

  2- on détermine le chiffre d'affaire par couple (region type) pour lequel il y a eu des ventes ;

  3- on détermine un chiffre d'affaire nul pour chaque couple (region type) pour lequel des vins existent mais aucune vente n'a eu lieu ;

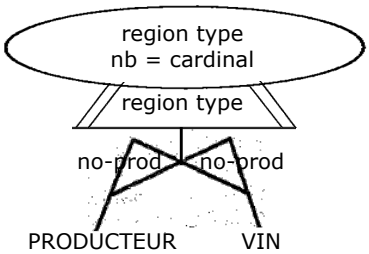
  4- on fait l'union des résultats 2 et 3 ;

  5- on réalise une jointure entre les résultats 1 et 4 de façon à associer à chaque couple (region type) pour lequel des vins existent le nombre de vins différents associés et le chiffre d'affaire correspondant, même nul.

- ✓ 1- déterminer le nombre de vins différents par couple (region type) ; pour cette partie, le cas 0 n'est pas pris en compte :

Req1 (region, type, nb)

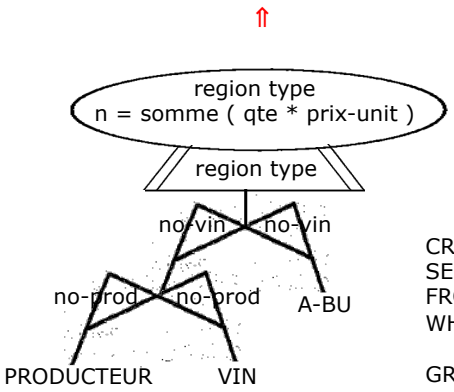
⇑




```
CREATE TEMPORARY TABLE req1
SELECT region, type, COUNT(*) nb
FROM producteur, vin
WHERE producteur.no-prod = vin.no-prod
GROUP BY region, type
```

✓ 2- on détermine le chiffre d'affaire par couple (region type) pour lequel il y a eu des ventes :

Req2 (region, type, n)



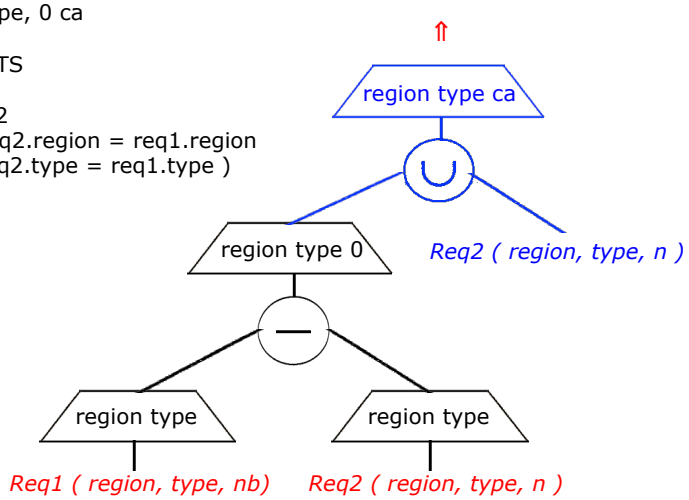

```
CREATE TEMPORARY TABLE req2
SELECT region, type, SUM(qte*prix-unit) n
FROM producteur, vin, a-bu
WHERE producteur.no-prod = vin.no-prod
   AND vin.no-vin = a-bu.no-vin
GROUP BY region, type
```

✓ 3- on détermine un chiffre d'affaire nul pour chaque couple (region type) pour lequel des vins existent mais aucune vente n'a eu lieu ;

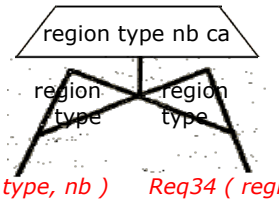
4- on fait l'union des résultats 2 et 3 :

```
CREATE TEMPORARY TABLE req34
SELECT region, type, 0 ca
FROM req1
WHERE NOT EXISTS
        ( SELECT *
          FROM req2
          WHERE req2.region = req1.region
             AND req2.type = req1.type )
      UNION
SELECT *
FROM req2
```

Req34 (region, type, ca)



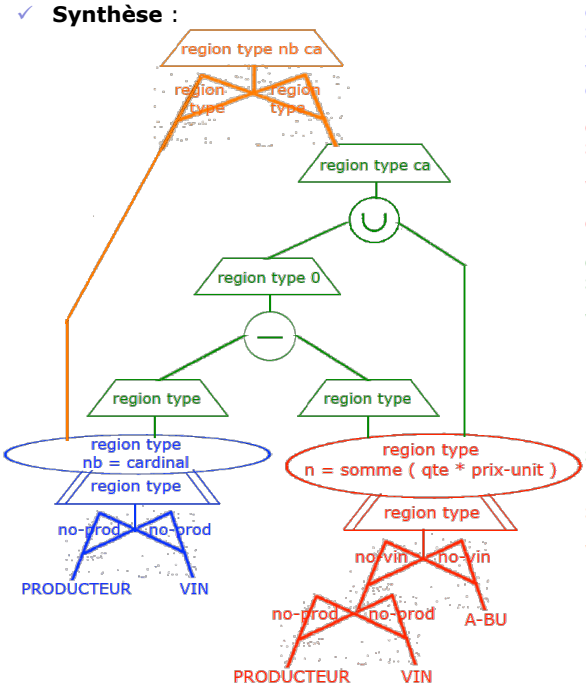
✓ 5- on réalise une jointure entre les résultats 1 et 4 de façon à associer à chaque couple (region type) pour lequel des vins existent le nombre de vins différents associés et le chiffre d'affaire correspondant, même nul.



Req1 (region, type, nb) *Req34 (region, type, ca)*

```
SELECT req1.region, type, nb, ca
FROM req1, req34
WHERE req1.region = req2.region
   AND req1.type = req2.type
;
DROP TABLE req1
;
DROP TABLE req2
;
DROP TABLE req34
```

✓ **Synthèse** :




```
CREATE TEMPORARY TABLE req1
SELECT region, type, COUNT(*) nb
FROM producteur, vin
WHERE producteur.no-prod = vin.no-prod
GROUP BY region, type
;
CREATE TEMPORARY TABLE req2
SELECT region, type, SUM(qte*prix-unit) n
FROM producteur, vin, a-bu
WHERE producteur.no-prod = vin.no-prod
   AND vin.no-vin = a-bu.no-vin
GROUP BY region, type
;
CREATE TEMPORARY TABLE req34
SELECT region, type, 0 ca
FROM req1
WHERE NOT EXISTS
          ( SELECT *
            FROM req2
            WHERE req2.region =
        req1.region
                AND req2.type = req1.type )
      UNION
SELECT *
FROM req1
;
SELECT req1.region, req2.type, nb, ca
FROM req1, req34
WHERE req1.region = req2.region
   AND req1.type = req2.type
;
DROP TABLE req1
;
DROP TABLE req2
;
DROP TABLE req34
```