3. Sous-requêtes simples

- **3.1 Sous-requêtes simples dans WHERE**
  - Cas 1 : 1 ligne / 1 colonne
  - Cas 2 : n lignes / 1 colonne
  - Cas 3 : 1 ligne / n colonnes
  - Cas 4 : m lignes / n colonnes
- **3.2 Sous-requêtes et jointures**
- **3.3 Sous-requêtes simples dans HAVING**
- **3.4 Sous-requêtes simples dans SELECT / ORDER BY**
- **3.5 Sous-requêtes simples dans FROM**

3.3 Sous-requêtes dans HAVING

- **HAVING = sélection des agrégats**
  - même principe que dans WHERE
  - condition du HAVING porte sur des calculs verticaux
  - ➔ ss-req qui produit une valeur / une liste de valeurs
    - souvent 1 colonne
    - souvent une requête qui donne le résultat d'un calcul vertical
      - sur une table ou sur des agrégats

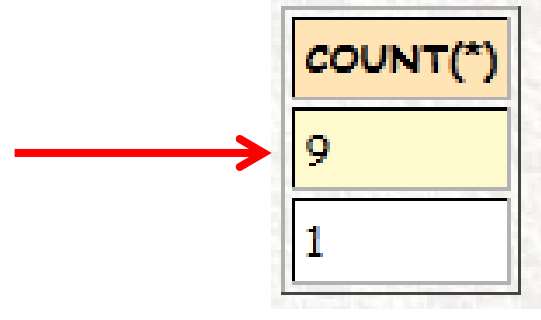
3.3 Sous-requêtes dans HAVING

- **Exemple**
  - Le genre dans lequel il y a le plus de films français

3.3 Sous-requêtes dans HAVING

- **Exemple**
  - Le genre dans lequel il y a le plus de films français
  - 1. calculer pour chaque genre le nombre de films français

```
SELECT COUNT(*)
FROM film
WHERE Fnat = 'FRANCE'
GROUP BY Fgenre
```



COUNT(*)
9
1

  - on voudrait calculer le max…
    …mais on ne peut pas enchaîner 2 calculs verticaux
    - ~~MAX(COUNT(*))~~
  - il faudrait passer par une table intermédiaire
    ➔ sous-requête dans le FROM… on verra plus loin !

3.3 Sous-requêtes dans HAVING

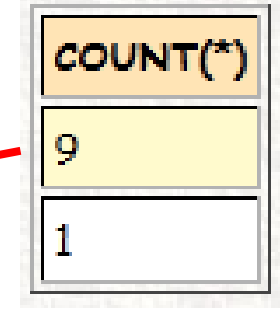
- **Exemple**
  - Le genre dans lequel il y a le plus de films français
  - 2. on cherche les genres dont le nombre de films est **supérieur ou égal à toutes** ces valeurs (opérateur `>= ALL`)
  -

```
SELECT Fgenre
FROM film
WHERE Fnat = 'FRANCE'
GROUP BY Fgenre
HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                        FROM film
                        WHERE Fnat = 'FRANCE'
                        GROUP BY Fgenre)
```

COUNT(*)
9
1

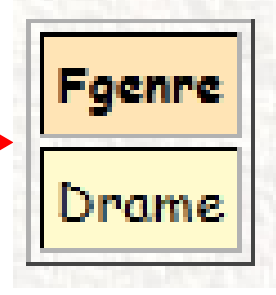
3.3 Sous-requêtes dans HAVING

- **Exemple**
  - Le genre dans lequel il y a le plus de films français
  - 2. on cherche les genres dont le nombre de films est **supérieur ou égal à toutes** ces valeurs (opérateur `>= ALL`)
  - ```
    SELECT Fgenre
    FROM film
    WHERE Fnat = 'FRANCE'
    GROUP BY Fgenre
    HAVING COUNT(*) >= ALL (9, 1)
    ```


    ```

3.3 Sous-requêtes dans HAVING

- **Exemple**
  - Le genre dans lequel il y a le plus de films français
  - 2. on cherche les genres dont le nombre de films est **supérieur ou égal à toutes** ces valeurs (opérateur `>= ALL`)
  -
```
SELECT Fgenre
FROM film
WHERE Fnat = 'FRANCE'
GROUP BY Fgenre
HAVING COUNT(*) >= ALL (9, 1)
```



Attention : on ne peut écrire la requête ainsi car `>= ALL` est un opérateur de liste qui ne fonctionne qu'avec une sous-requête…

3.3 Sous-requêtes dans HAVING

- **Exemple**
  - Le genre dans lequel il y a le plus de films français

  - ```
    SELECT Fgenre
    FROM film
    WHERE Fnat = 'FRANCE'
    GROUP BY Fgenre
    HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                            FROM film
                            WHERE Fnat = 'FRANCE'
                            GROUP BY Fgenre)
    ```
    ```

3. Sous-requêtes simples

- **3.1 Sous-requêtes simples dans WHERE**
  - Cas 1 : 1 ligne / 1 colonne
  - Cas 2 : n lignes / 1 colonne
  - Cas 3 : 1 ligne / n colonnes
  - Cas 4 : m lignes / n colonnes
- **3.2 Sous-requêtes et jointures**
- **3.3 Sous-requêtes simples dans HAVING**
- → **3.4 Sous-requêtes simples dans SELECT / ORDER BY**
- **3.5 Sous-requêtes simples dans FROM**

3.4.1 Sous-requêtes dans SELECT

- **`SELECT` : liste d'attributs (à afficher)**
  - … comme GROUP BY
    - dans lequel on ne peut pas mettre de sous-requête…
      … car ss-req renvoie des valeurs, pas des attributs

➔ pourquoi peut-on mettre des ss-req dans **`SELECT`** ?
  - et où ?

3.4.1 Sous-requêtes dans SELECT

- **`SELECT` : liste d'attributs (à afficher)**
  - … comme GROUP BY
    - dans lequel on ne peut pas mettre de sous-requête…
      … car ss-req renvoie des valeurs, pas des attributs

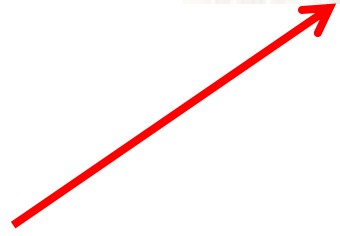
  ➔ pourquoi peut-on mettre des ss-req dans **`SELECT`** ?
    - et où ?

- **`SELECT` peut contenir des *champs calculés***
  - ss-req dont le résultat participe au calculs
    - ➔ 1 valeur = 1 ligne / 1 colonne

3.4.1 Sous-requêtes dans SELECT

- **Exemple**
  - Pour chaque film, la moyenne de ses notes, et la différence entre cette moyenne et la moyenne de toutes les notes

AVG(Nnote)
11.5967

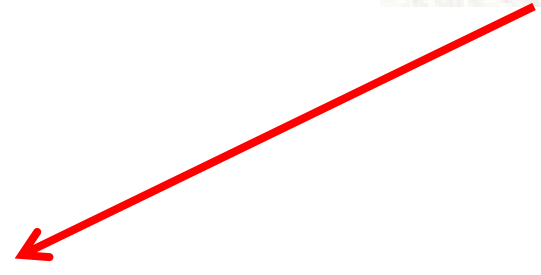


```
SELECT Ftitre, AVG(Nnote),
       AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
FROM film JOIN note ON FIlmID = NFilmID
GROUP BY FilmID, Ftitre
```

3.4.1 Sous-requêtes dans SELECT

- **Exemple**
  - Pour chaque film, la moyenne de ses notes, et la différence entre cette moyenne et la moyenne de toutes les notes

AVG(Nnote)
11.5967



```
SELECT Ftitre, AVG(Nnote),
       AVG(Nnote) - (11.5967)
FROM film JOIN note ON FIlmID = NFilmID
GROUP BY FilmID, Ftitre
```

3.4.1 Sous-requêtes dans SELECT

- **Exemple**
  - Pour chaque film, la moyenne de ses notes, et la différence entre cette moyenne et la moyenne de toutes les notes

Ftitre	AVG(Nnote)	AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
Million dollar baby	12.8636	1.2670
Eyes wide shut	14.2381	2.6414
Le bon, la brute et le truand	12.2727	0.6760
Les oiseaux	12.6250	1.0283
Psychose	13.8182	2.2215
Psycho	10.3704	-1.2263
Hero	9.0400	-2.5567
Hana-bi	12.6087	1.0120
L'homme qui en savait trop	13.2609	1.6642
Les misérables	14.0000	2.4033
L'homme qui en savait trop	8.8667	-2.7300
Les misérables	7.7083	-3.8883
Les misérables	10.1600	-1.4367
S'en fout la mort	10.1667	-1.4300

```
SELECT Ftitre, AVG(Nnote),
       AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
FROM film JOIN note ON FIlmID = NFilmID
GROUP BY FilmID, Ftitre
```

3.4.2 Sous-requêtes dans ORDER BY

- **`ORDER BY` : liste d'attributs**
  - … comme SELECT !
- **`ORDER BY` peut contenir des *champs calculés***
  - ss-req dont le résultat participe au calculs
    - ➔ 1 valeur = 1 ligne / 1 colonne

3.4.2 Sous-requêtes dans ORDER BY

- **Exemple**
  - Pour chaque film, la moyenne de ses notes, et la différence entre cette moyenne et la moyenne de toutes les notes ; classer les résultats selon cette différence

```
SELECT Ftitre, AVG(Nnote),
      AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
FROM film JOIN note ON FIlmID = NFilmID
GROUP BY FilmID, Ftitre
ORDER BY AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
```

  - Remarque : intérêt limité ici… même résultat avec
    - `ORDER BY AVG(Nnote)`

3.4.2 Sous-requêtes dans ORDER BY

- Faut-il recopier la formule ?

```
SELECT Ftitre, AVG(Nnote),
     AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
FROM film JOIN note ON FIlmID = NFilmID
GROUP BY FilmID, Ftitre
ORDER BY AVG(Nnote) - (SELECT AVG(Nnote) FROM note)
```

- … ou peut-on utiliser un alias et écrire :

```
SELECT Ftitre, AVG(Nnote),
     AVG(Nnote) - (SELECT AVG(Nnote) FROM note) AS diff
FROM film JOIN note ON FIlmID = NFilmID
GROUP BY FilmID, Ftitre
ORDER BY diff
```

- norme SQL : recopier !
- MySQL : on peut utiliser l'alias, mais… nom de l'alias, bug…

3. Sous-requêtes simples

- **3.1 Sous-requêtes simples dans WHERE**
  - Cas 1 : 1 ligne / 1 colonne
  - Cas 2 : n lignes / 1 colonne
  - Cas 3 : 1 ligne / n colonnes
  - Cas 4 : m lignes / n colonnes
- **3.2 Sous-requêtes et jointures**
- **3.3 Sous-requêtes simples dans HAVING**
- **3.4 Sous-requêtes simples dans SELECT / ORDER BY**
- **3.5 Sous-requêtes simples dans FROM**

3.5 Sous-requêtes dans FROM

- **`FROM` : table de travail de la requête**

- **Ss-req dans `FROM`**
  - table de travail obtenue par une sous-requête
  - ➔ table intermédiaire

  - il faut nommer la table obtenue par sous requête (AS)
  - noms des colonnes de cette table utilisables dans req. princip.

  - souvent utilisées pour « enchaîner » 2 calculs verticaux

3.5 Sous-requêtes dans FROM

- **Exemple**
  - Le genre dans lequel il y a le plus de films français

```
SELECT Fgenre
FROM film
WHERE Fnat = 'FRANCE'
GROUP BY Fgenre
HAVING COUNT(*) = (SELECT MAX(nb)
                   FROM (SELECT COUNT(*) AS nb
                         FROM film
                         WHERE Fnat = 'FRANCE'
                         GROUP BY Fgenre) AS tab)
```

3.5 Sous-requêtes dans FROM

- **Exemple**
  - Le genre dans lequel il y a le plus de films français

```
SELECT Fgenre
FROM film
WHERE Fnat = 'FRANCE'
GROUP BY Fgenre
HAVING COUNT(*) = (SELECT MAX(nb)
                   FROM (SELECT COUNT(*) AS nb
                         FROM film
                         WHERE Fnat = 'FRANCE'
                         GROUP BY Fgenre) AS tab)
```

Colonne de la table produite par la sous-requête

Renommage de la table produite par la sous-requête

3.5 Sous-requêtes dans FROM

- **Exemple**
  - Le genre dans lequel il y a le plus de films français

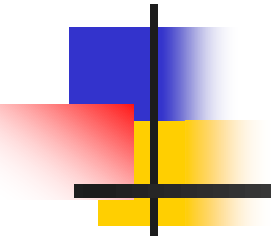
Fgenre
Drame

```
SELECT Fgenre
FROM film
WHERE Fnat = 'FRANCE'
GROUP BY Fgenre
HAVING COUNT(*) = (SELECT MAX(nb)
                   FROM (SELECT COUNT(*) AS nb
                         FROM film
                         WHERE Fnat = 'FRANCE'
                         GROUP BY Fgenre) AS tab)
```

MAX(nb)
9

tab

nb
9
1



Sommaire

4. Sous-requêtes corrélées

- **Corrélation**
  - = dépendance entre requêtes interne et externe
    - le résultat de la requête interne dépend …
      … de la ligne en cours d'évaluation de la requête externe
  - ➔ req. interne évaluée **pour chaque ligne** de la req. externe

- On dit aussi **requête synchronisée**

- **Pouvoir expressif**
  - Requête = 1 boucle sur les lignes d'une table
  - Requête + sous-requête corrélée = 2 boucles imbriquées !

4. Sous-requêtes corrélées

- **Élément de corrélation dans requête interne**
  - Souvent 1 référence à 1 colonne de la requête externe

  - ```
    SELECT ...
    FROM table1 AS T1 ...
    WHERE ... (SELECT
                FROM ...
                WHERE T1.col1 ...);
    ```

  - résultat de la requête interne dépend de la valeur T1.col1…
    … donc de la ligne en cours d'évaluation de la requête externe

  ➔ 1 exécution de la ss-req. pour chaque ligne de T1
    ```

4. Sous-requêtes corrélées

- **Position de la sous-requête corrélée**
  - partout où on peut trouver une sous-requête (norme SQL)
  - attention aux limitations des SGBD…
  - Opérateur spécifique aux ss-req corrélées (WHERE/HAVING)
    - opérateur **`EXISTS`** (ou **`NOT EXISTS`**)
      - (non-) existence d'un résultat à la requête interne pour la valeur courante de la requête externe
    - opérateur **`UNIQUE`** (ou **`NOT UNIQUE`**)
      - (non-) unicité d'un résultat à la requête interne
    - ➔ SELECT * dans ce cas (car résultat lui-même pas important)

4. Sous-requêtes corrélées

- **Exemple**
  - Nom, prénom et âge des réalisateurs lors de la sortie de leur premier film

  - ```
    SELECT Anom, Aprenom,
                  (SELECT MIN(Fannee)
                   FROM film
                   WHERE FrealisateurID = ArtisteID)
                  - Anaissance
    FROM artiste
    WHERE ArtisteID IN
          (SELECT DISTINCT FrealisateurID
           FROM film)
    ```
    ```

4. Sous-requêtes corrélées

- **Exemple**
  - Nom, prénom et âge des réalisateurs lors de la sortie de leur premier film

```
SELECT Anom, Aprenom,
       (SELECT MIN(Fannee)
        FROM film
        WHERE FrealisateurID = ArtisteID)
       - Anaissance
FROM artiste
WHERE ArtisteID IN
      (SELECT DISTINCT FrealisateurID
       FROM film)
```

extrait les réalisateurs de la table artiste
➔ hors propos !

4. Sous-requêtes corrélées

plus petite année de sortie d'un film de « ArtisteID »

- **Exemple**
  - Nom, prénom et âge des réalisateurs lors de la sortie de leur premier film

```
SELECT Anom, Aprenom,
                (SELECT MIN(Fannee)
                 FROM film
                 WHERE FrealisateurID = ArtisteID)
                - Anaissance
FROM artiste
WHERE ArtisteID IN
        (SELECT DISTINCT FrealisateurID
         FROM film)
```

4. Sous-requêtes corrélées

- **Exemple**
  - les membres qui n'ont pas noté de film anglais

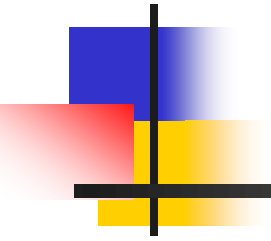
  - ```
    SELECT Mnom, Mprenom
    FROM membre
    WHERE NOT EXISTS
          (SELECT *
           FROM film JOIN note ON FilmID = NFilmID
           WHERE Fnat = 'UK' AND NmembreID = MembreID)
    ```
    ```

4. Sous-requêtes corrélées

- **Exemple**
  - les membres qui n'ont pas noté de film anglais

les films anglais notés par MembreID

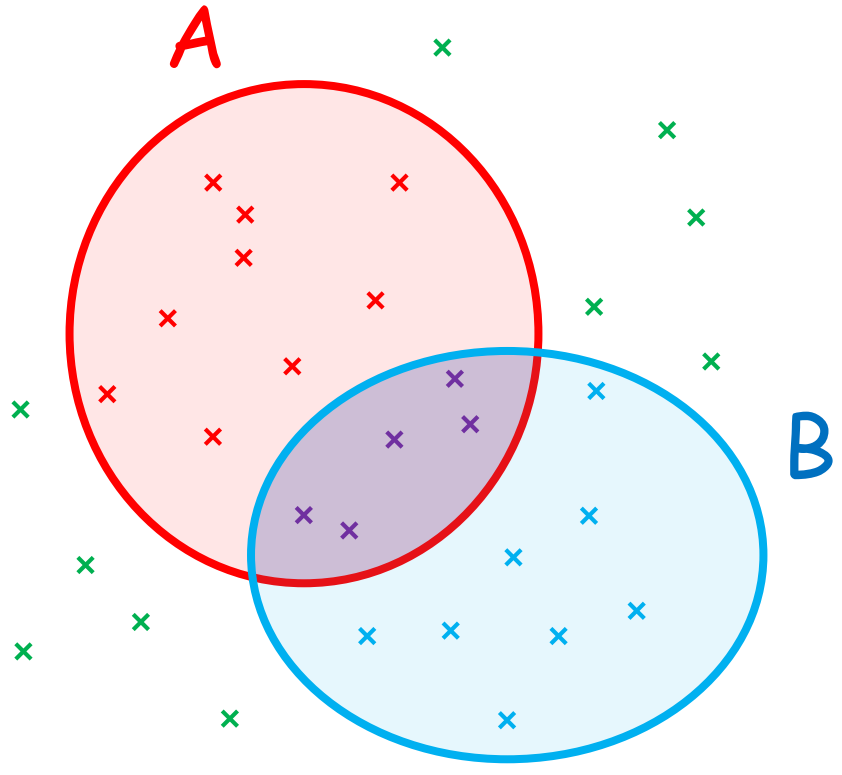
```
SELECT Mnom, Mprenom
FROM membre
WHERE NOT EXISTS
      (SELECT *
       FROM film JOIN note ON FilmID = NFilmID
       WHERE Fnat = 'UK' AND NmembreID = MembreID)
```



Sommaire

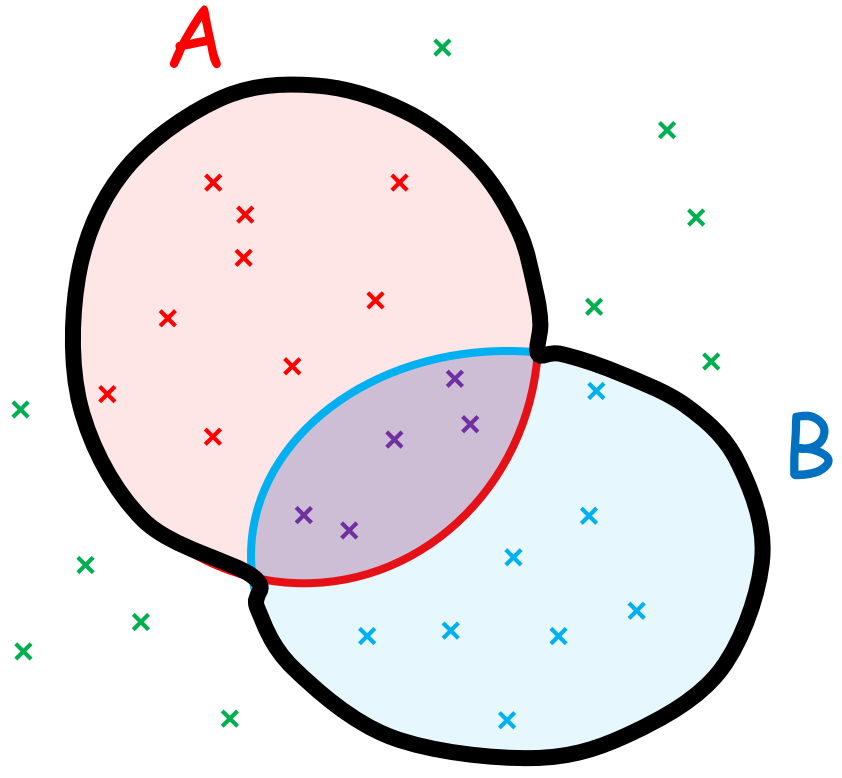
5. Opérations ensemblistes

- 5.1 Union
- 5.2 Intersection
- 5.3 Différence



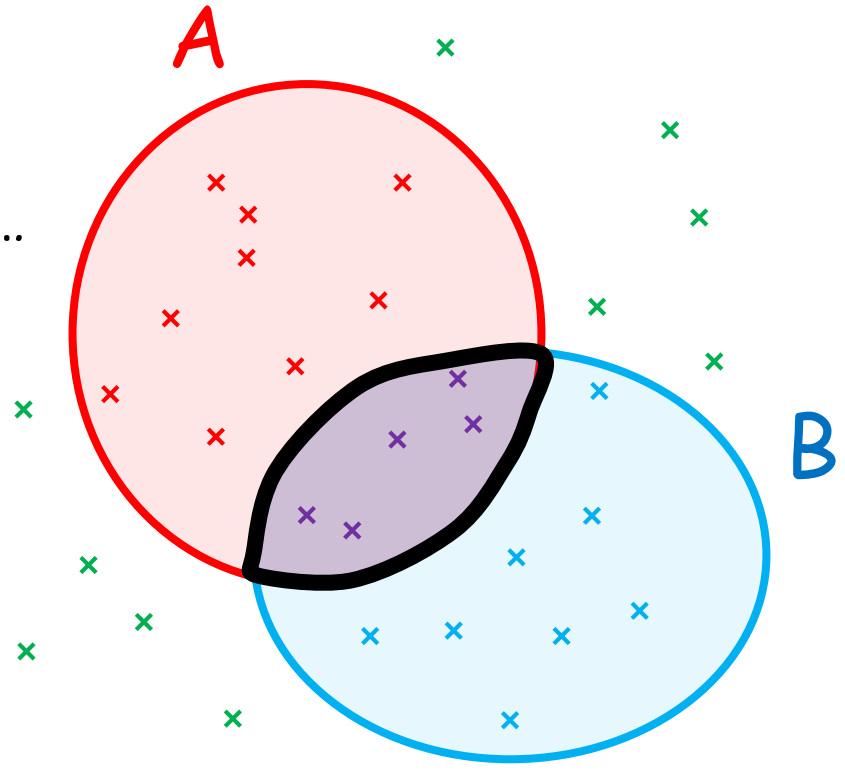
5. Opérations ensemblistes - rappels

- **Union**
  - $A \cup B$
  - éléments dans A ou dans B
  - ex : les croix
    - rouges + violettes
    - bleues + violettes
    - (pas les vertes !)
    - *rem : violettes x1 ou x2*



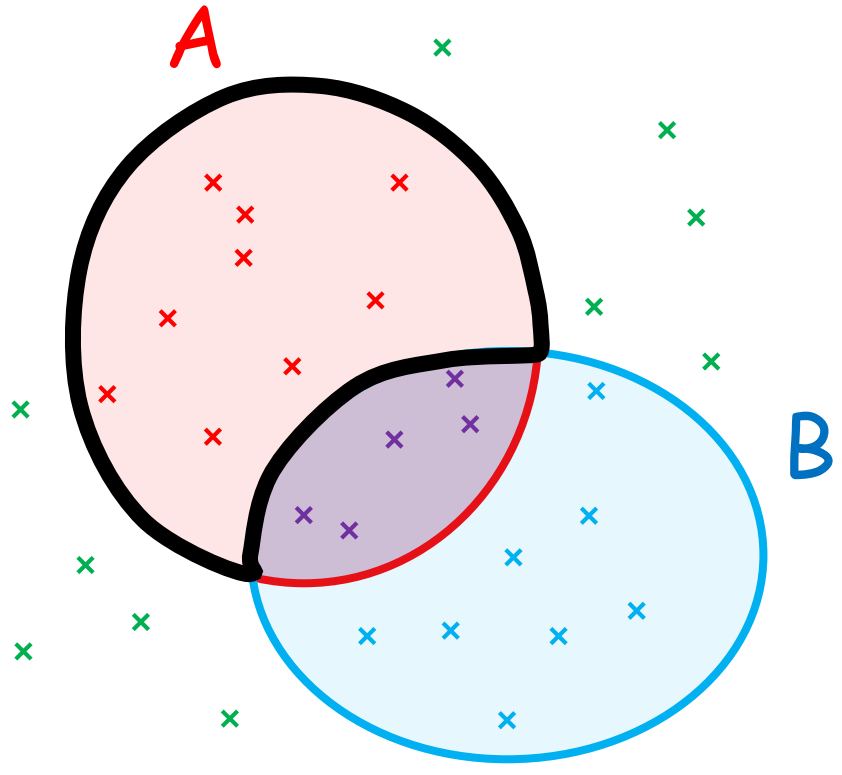
5. Opérations ensemblistes - rappels

- **Intersection**
  - $A \cap B$
  - éléments dans A et dans B…
    - … à la fois !
  - ex : les croix
    - violettes
    - *rem : violettes x1 ou x2*



5. Opérations ensemblistes - rappels

- **Différence**
  - A \ B
  - éléments dans A
    mais pas dans B…
  - ex : les croix
    - rouges
  - *rem :* $A \setminus B \neq B \setminus A$
    (B \ A : croix bleues)



5.1 Union

- **Rassemble les résultats de 2 requêtes**
  - les 2 résultats doivent être compatibles
    - même nombre de colonnes
    - même ordre des colonnes (avec domaines compatibles)

- **Syntaxe**
  - ```
    <requête 1>
    UNION [ALL | DISTINCT | ε]
    <requête 2> ;
    ```
  - Par défaut DISTINCT = supprime les doublons
    - ➔ on écrit `UNION` ou `UNION ALL`
    ```

5.1 Union

- **Ordonner les réponses**
  - 1 seul ORDER BY autorisé
  - placé après la dernière requête (agit sur l'union)
  - attributs de la première requête ou position

- **Union de plus de 2 requêtes**
  - ```
    <requête 1>
    UNION [ALL | DISTINCT | ε]
    <requête 2>
    UNION [ALL | DISTINCT | ε]
    ...
    UNION [ALL | DISTINCT | ε]
    <requête n>
    ```
  - parenthèses éventuelles pour forcer l'ordre
    - si mélange de ALL et de DISTINCT
    ```

5.1 Union

- **Exemple**
  - les années de naissance et de décès des artistes

```
SELECT Anaissance AS années
FROM artiste

UNION

SELECT Amort AS années
FROM artiste
WHERE Amort IS NOT NULL

ORDER BY années
```

années
1880
1889
1891
1892
1893
1897
1899
1900
1904
1908
1909
1910
1913
1915
1916
1917
1919

5.2 Intersection

- **Données communes aux résultats de 2 requêtes**
  - les 2 résultats doivent être compatibles
    - même nombre de colonnes
    - même ordre des colonnes (avec domaines compatibles)

- **Syntaxe**
  - ```
    <requête 1>
    INTERSECT [ALL | DISTINCT | ε]
    <requête 2> ;
    ```
  - par défaut `DISTINCT` = supprime les doublons
    - ➔ on écrit `INTERSECT` ou `INTERSECT ALL`
    ```

5.2 Intersection

- **Ordonner les réponses**
  - 1 seul ORDER BY autorisé
  - placé après la dernière requête (agit sur l'intersection)

- **Intersection de plus de 2 requêtes**
  - ```
    <requête 1>
    INTERSECT [ALL | DISTINCT | ε]
    <requête 2>
    INTERSECT [ALL | DISTINCT | ε]
    ...
    INTERSECT [ALL | DISTINCT | ε]
    <requête n>
    ```
  - parenthèses éventuelles pour forcer l'ordre
    - si mélange de ALL et de DISTINCT
    ```

5.2 Intersection

- **Exemple**
  - Les artistes qui ont réalisé des films anglais *et* des films américains
  - = ceux qui sont à la fois dans
    - l'ensemble des réalisateurs de films anglais
    - l'ensemble des réalisateurs de films américains

```
SELECT Anom, Aprenom
FROM artiste JOIN film ON ArtisteID = FrealisateurID
WHERE Fnat = 'UK'
INTERSECT
SELECT Anom, Aprenom
FROM artiste JOIN film ON ArtisteID = FrealisateurID
WHERE Fnat = 'USA'
```

5.2 Intersection

- **INTERSECT pas disponible avec MySQL**
  - mais on peut s'en passer…
  - fait avec une sous-requête liée avec IN

```
SELECT DISTINCT Anom, Aprenom
FROM artiste JOIN film ON ArtisteID = FrealisateurID
WHERE Fnat = 'UK'
  AND (Anom, Aprenom) IN (SELECT DISTINCT (Anom, Aprenom)
                          FROM artiste JOIN film ON
                                  ArtisteID = FrealisateurID
                          WHERE Fnat = 'USA')
```

5.2 Intersection

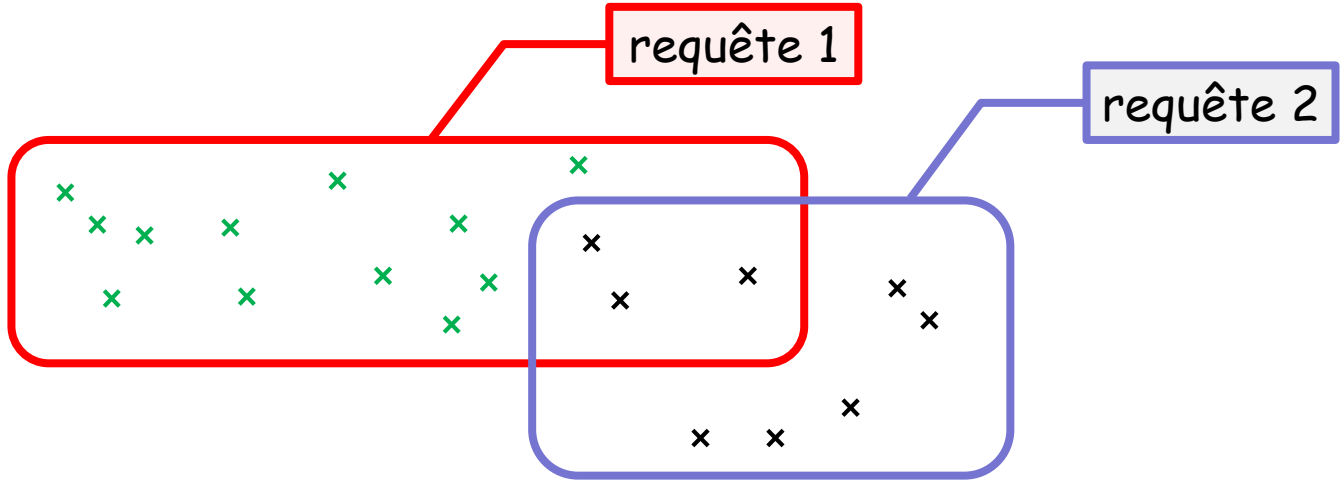
- **Remarque**
  - capture les homonymes (un USA et l'autre UK ➔ les 2… à tort)
  - avec IN/ss-req plutôt qu' INTERSECT, on peut faire mieux

```
SELECT DISTINCT Anom, Aprenom
FROM artiste JOIN film ON ArtisteID = FrealisateurID
WHERE Fnat = 'UK'
  AND ArtisteID IN (SELECT DISTINCT ArtisteID
                    FROM artiste JOIN film ON
                                 ArtisteID = FrealisateurID
                    WHERE Fnat = 'USA')
```

5.3 Différence

- **Différence ensembliste des résultats de 2 requêtes**
  - `<requête 1>  EXCEPT  <requête 2>`
  - = les résultats de la req. 1 qui ne sont pas dans la req. 2
  - = requête 1 moins l'intersection des deux

Les bonnes réponses sont les croix vertes !



5.3 Différence

- **Différence ensembliste des résultats de 2 requêtes**
  - les 2 résultats doivent être compatibles
    - même nombre de colonnes
    - même ordre des colonnes (avec domaines compatibles)
- **Syntaxe**
  - ```
    <requête 1>
    EXCEPT
    <requête 2>
    ```
  - **Attention** : opération non commutative !
  - noté `MINUS` dans certains SGBD (e.g. Oracle)
    ```

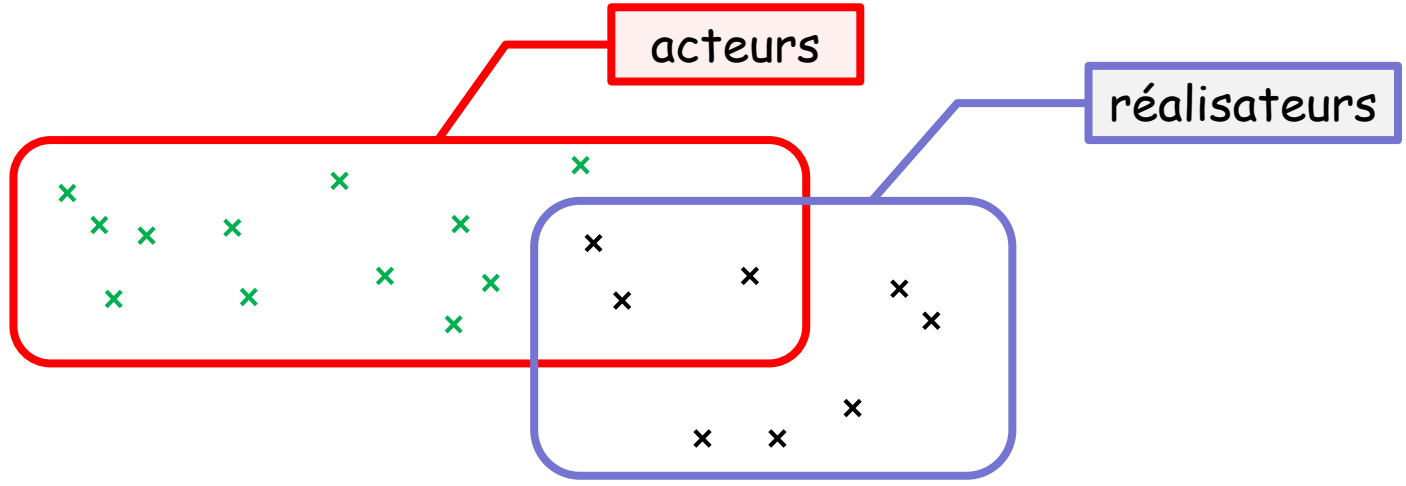
5.3 Différence

- **Ordonner les réponses**
  - 1 seul ORDER BY autorisé
  - placé après la dernière requête (agit sur la différence)
- **Différence ensembliste de plus de 2 requêtes**
  - ```
    <requête 1>
    EXCEPT
    <requête 2>
    EXCEPT
    ...
    EXCEPT
    <requête n>
    ```
  - parenthèses éventuelles pour forcer l'ordre
    ```

5.3 Différence

- **Exemple**
  - les acteurs qui ne sont pas réalisateurs

Les bonnes réponses sont les croix vertes !



5.3 Différence

- **Exemple**
  - les acteurs qui ne sont pas réalisateurs

```
SELECT DISTINCT Anom, Aprenom
FROM artiste JOIN joue ON ArtisteID = JArtisteID

EXCEPT

SELECT DISTINCT Anom, Aprenom
FROM artiste JOIN film ON ArtisteID = FrealisateurID
```

5.3 Différence

- **EXCEPT pas disponible avec MySQL**
  - mais on peut s'en passer…
  - fait avec une sous-requête liée avec `NOT IN`

```
SELECT DISTINCT Anom, Aprenom
FROM artiste JOIN joue ON ArtisteID = JArtisteID
WHERE ArtisteID NOT IN
              (SELECT DISTINCT ArtisteID
               FROM artiste JOIN film ON
                                  ArtisteID = FrealisateurID)
```

  - lien sur `ArtisteID` plutôt que `(Anom, Aprenom)` cf. précédemment

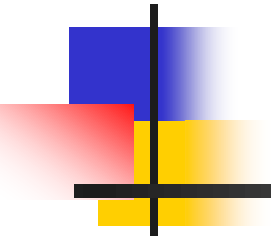
5.3 Différence

- **EXCEPT pas disponible avec MySQL**
  - mais on peut s'en passer…
  - fait avec une sous-requête liée avec `NOT IN`

```
SELECT DISTINCT Anom, Aprenom
FROM artiste JOIN joue ON ArtisteID = JArtisteID
WHERE ArtisteID NOT IN
                (SELECT DISTINCT ArtisteID
                 FROM artiste JOIN film ON
                                ArtisteID = FrealisateurID)
```

acteurs

réalisateurs



Sommaire

6.1 Division de relations

- **Ce qu'on cherche**
  - dividende `R(A,B)` diviseur `S(B)` quotient `T(A)`
  - division de relations

    $T(A) = R(A,B) \div S(B)$

    $T(A) \times S(B) \subseteq R(A,B)$ où $T(A)$ est maximal
- **Ce que ça représente**
  - ***tous*** les « A » qui sont en relation avec tous « B » de S dans R
    i.e. tous les « A » tels que « A x B » $\in$ R pour tous les « B »
  - exemple : « quels sont les membres du club qui ont noté tous les films français ? »

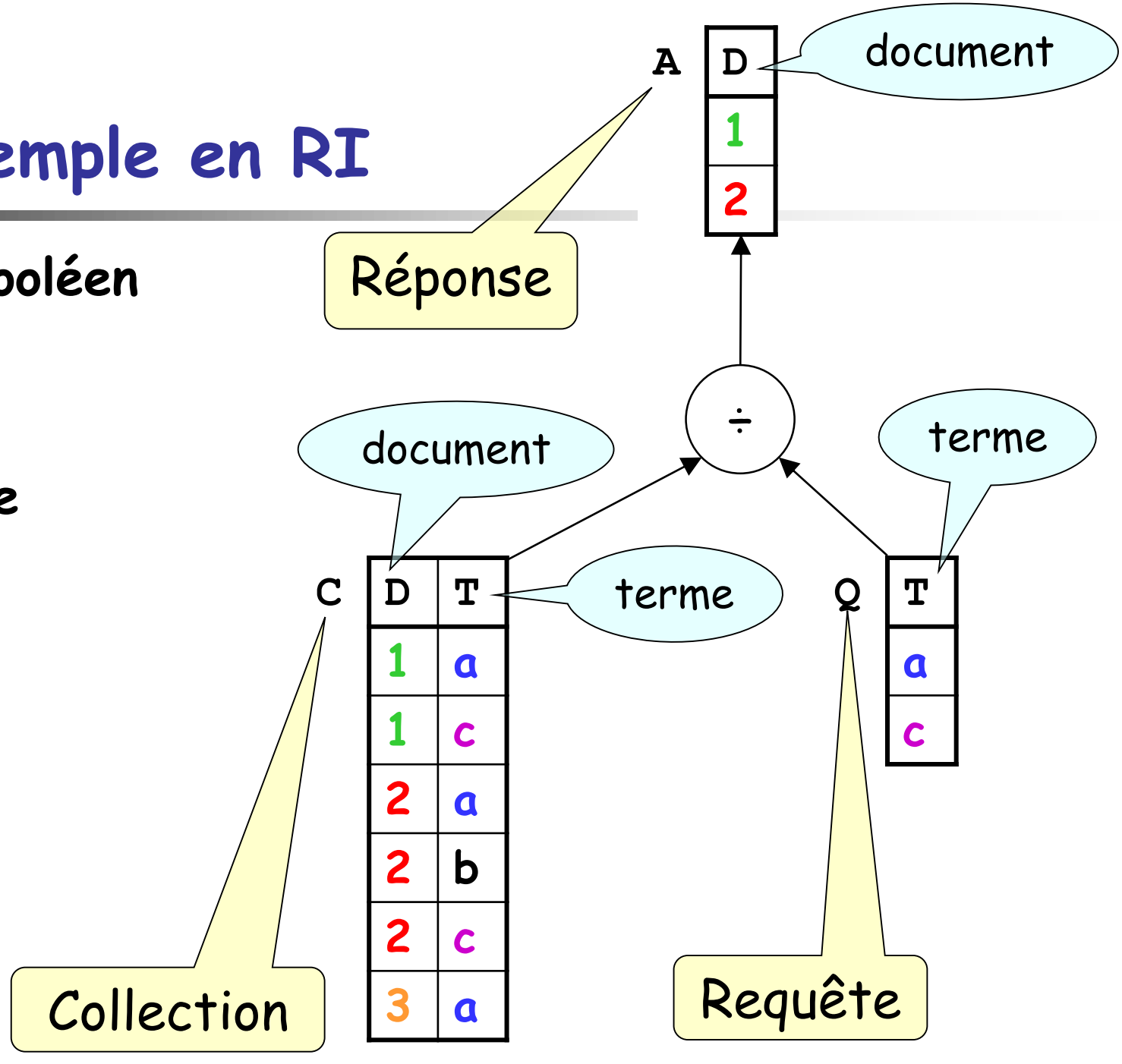
6.2 Exemple en recherche d'information

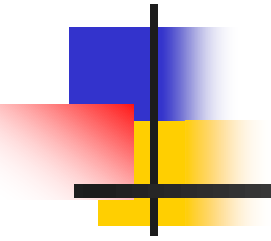
Quel est l'ensemble `A` des documents réponse
contenant les documents `d` de la collection `C`
contenant tous les termes `t` de la requête `Q`

- **En RI textuelle**
  - = modèle booléen de RI (ancêtre des moteurs de recherche)
- **En BD**
  - Collection : relation `C(d,t)` Requête : relation `Q(t)`
  - = division de relations `A = C ÷ Q`
    `A × Q ⊆ C` où `A` est maximal

6.2 Exemple en RI

Modèle booléen
de RI
=
division de
Relations
en BD





6.3. Division en SQL

- **Opération absente de SQL**
  - Mais bien utile parfois…
  - Plusieurs façons d'écrire une division en SQL
    - la division n'est pas une opération de base
  - Solution à éviter : comptage
    - peut fonctionner
    - mais il faut compter les bonnes choses ➔ requêtes complexes !

6.3. Division en SQL

- **Une solution « simple »**
  - Utilise une double requête corrélée
  - Division de R(A, B) par S(B) donne T(A)
    $T(A) = R \div S = \{a \mid a \in Proj_A(R) \text{ et } \forall b \in S, (a, b) \in R\}$
    peut se réécrire avec une double négation
    $T(A) = R \div S = \{a \mid a \in Proj_A(R) \text{ et } \nexists b \in S, (a, b) \notin R\}$
  - Ex : « quels sont les membres du club qui ont noté tous les films français ? »
    ➔ « quels sont les membres du club pour lesquels il n'existe pas de film français qu'ils n'aient pas noté ? »

6.3. Division en SQL

- Division de R(A,B) par S(B)

```
SELECT A
FROM T AS t
WHERE NOT EXISTS
        (SELECT *
         FROM S AS s
         WHERE NOT EXISTS
                 (SELECT *
                  FROM R
                  WHERE A = t.A AND B = s.B))
```

6.4. Exemple de division en SQL

« quels sont les membres du club pour lesquels il n'existe pas de film français qu'ils n'aient pas noté ? »

```
SELECT Mnom, Mprenom
FROM membre AS m
WHERE NOT EXISTS
      (SELECT *
       FROM film AS f
       WHERE Fnat = 'FRANCE'
         AND NOT EXISTS
             (SELECT *
              FROM note
              WHERE NmembreID = m.membreID
                AND NfilmID = f.filmID));
```

6.4. Exemple de division en SQL

« quels sont les membres du club pour lesquels il n'existe pas de film français qu'ils n'aient pas noté ? »

```
SELECT Mnom, Mprenom
FROM membre AS m
WHERE NOT EXISTS
        (SELECT *
         FROM film AS f
         WHERE Fnat = 'FRANCE'
           AND NOT EXISTS
                (SELECT *
                 FROM note
                 WHERE NmembreID = m.membreID
                   AND NfilmID = f.filmID));
```

Note mise
par le membre exploré
au film français exploré

6.4. Exemple de division en SQL

« quels sont les membres du club pour lesquels il n'existe pas de film français qu'ils n'aient pas noté ? »

```
SELECT Mnom, Mprenom
FROM membre AS m
WHERE NOT EXISTS
    (SELECT *
     FROM film AS f
     WHERE Fnat = 'FRANCE'
       AND NOT EXISTS
           (SELECT *
            FROM note
            WHERE NmembreID = m.membreID
              AND NfilmID = f.filmID));
```

Ensemble des films français pour lesquels il n'existe pas de note que le membre exploré ait mise (= qu'il n'a pas noté)

Note mise par le membre exploré au film français exploré

6.4. Exemple de divisio

Ensemble des
membres du club
pour lesquels il n'existe pas
de film français qu'il n'aient pas noté

« quels sont les membres du club pour lesquels il n'existe pas de film français qu'ils n'aient pas noté ? »

```
SELECT Mnom, Mprenom
FROM membre AS m
WHERE NOT EXISTS
      (SELECT *
       FROM film AS f
       WHERE Fnat = 'FRANCE'
         AND NOT EXISTS
             (SELECT *
              FROM note
              WHERE NmembreID = m.membreID
                AND NfilmID = f.filmID));
```

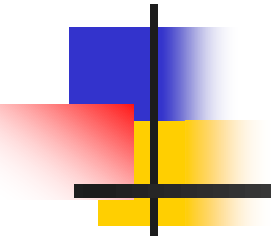
Note mise
par le membre exploré
au film français exploré

Ensemble des
films français pour lesquels
il n'existe pas de note
que le membre exploré
ait mise (= qu'il n'a pas noté)

6.4. Exemple de division en SQL

- **Attention : les relations ont été simplifiées**
- **Dividende R(A, B)**
  - jointure de film, note, membre
    - A : les membres qui ont mis des notes
      - membreID (+ tous les champs en dépendance fonctionnelle)
    - B : les film français
      - filmID (+ tous les champs en dépendance fonctionnelle)

  ➔ table réduite à note (qui donne le lien entre A et B)
- **Diviseur S(B)**
  - jointure de film, note ➔ réduit à film (français)
- **Quotient T(A)**
  - jointure de membre, note ➔ réduit à membre



Questions ?