

# Manipulation et interrogation de données (SQL DML)

## BD-BioInfo Bases de données

Cristina Sirangelo  
IRIF, Université Paris Diderot  
[cristina@irif.fr](mailto:cristina@irif.fr)

# Interrogation de données en SQL

# Interrogation de bases de données

Interroger une base de données: “extraire” des données de la base (sous forme d’une nouvelle table)

Requête: spécifie la nouvelle table à partir des tables dans la base

Exemples de requêtes:

quelles sont tous les films de 1954 ?

quels réalisateurs ont réalisé le plus de films?

SQL permet d’exprimer les requêtes de façon **déclarative**

on exprime **quelles sont les données** qu’on veut extraire

**PAS comment les extraire**

# Interroger une table

```
SELECT * FROM Film;
```

sélectionne toutes les colonnes (\*) de la table Film

Resultat :

| titre        | annee | realisateur |
|--------------|-------|-------------|
| Alien        | 1979  | Scott       |
| Vertigo      | 1958  | Hitchcock   |
| Psychose     | 1960  | Hitchcock   |
| Kagemusha    | 1980  | Kurosawa    |
| Volte-face   | 1997  | Woo         |
| Pulp Fiction | 1995  | Tarantino   |
| Titanic      | 1997  | Cameron     |
| Sacrifice    | 1986  | Tarkovski   |
| Match Point  | NULL  | Allen       |

# Interroger une table

On peut sélectionner les colonnes de son choix

```
SELECT titre, année FROM Film;
```

Resultat :

| titre        | annee |
|--------------|-------|
| Alien        | 1979  |
| Vertigo      | 1958  |
| Psychose     | 1960  |
| Kagemusha    | 1980  |
| Volte-face   | 1997  |
| Pulp Fiction | 1995  |
| Titanic      | 1997  |
| Sacrifice    | 1986  |
| Match Point  | NULL  |

# Interroger une table

La clause **WHERE** permet de définir un ensemble de conditions qui doivent être vérifiées par les lignes retenues

```
SELECT titre, annee  
FROM Film  
WHERE annee >= 1995;
```

Resultat :

| titre        | annee |
|--------------|-------|
| Volte-face   | 1997  |
| Pulp Fiction | 1995  |
| Titanic      | 1997  |

Sélectionne les titres et les années des films dont l'année est supérieure ou égale à 1995

# Requêtes SQL : forme générique

- Forme générique d'une requête :

SELECT <liste attributs>

FROM <liste tables>

WHERE <condition>

- Au delà des requêtes de base : plusieurs options pour les clauses SELECT, FROM et WHERE

# Clause WHERE : suivie d'une condition

**Condition simple** attribut opérateur valeur **ou**

attribut opérateur attribut

## opérateur

## exemple

|                                 |                     |                                      |
|---------------------------------|---------------------|--------------------------------------|
| <u>égalité</u>                  | =                   | <u>nom = 'Alice'</u>                 |
| <u>inégalité</u>                | <>                  | <u>article &lt;&gt; 'pull'</u>       |
| <u>inférieur ou égal</u>        | <=                  | <u>date_naiss &lt;= '1990-01-01'</u> |
| <u>inférieur</u>                | <                   |                                      |
| <u>supérieur</u>                | >                   |                                      |
| <u>supérieur ou égal</u>        | >=                  |                                      |
| <u>intervalle</u>               | BETWEEN ... AND ... | <u>prix between 100 and 200</u>      |
| <u>null test</u>                | is NULL             | <u>date_naiss is NULL</u>            |
| <u>reconnaissance de motifs</u> | LIKE '...'          | <u>nom LIKE 'Jul%'</u>               |




# Clause WHERE : BETWEEN

```
SELECT titre, annee
FROM Films
WHERE titre BETWEEN 'Psychose' AND 'Titanic';
```

Films

| titre        | annee | realisateu |
|--------------|-------|------------|
| Alien        | 1979  | 1          |
| Vertigo      | 1958  | 2          |
| Psychose     | 1960  | 2          |
| Kagemusha    | 1980  | 3          |
| Volte-face   | 1997  | 4          |
| Pulp Fiction | 1995  | 5          |
| Titanic      | 1997  | 6          |
| Sacrifice    | 1986  | 7          |



|              |       |
|--------------|-------|
| titre        | annee |
| Pulp Fiction | 1995  |
| Psychose     | 1960  |
| Titanic      | 1997  |
| Sacrifice    | 1986  |

# Clause WHERE : LIKE

- A LIKE 'motif'  
vrai si la valeur de A est conforme au motif
- 'motif': une chaîne de caractères qui peut inclure les caractères spéciaux "" et "%"
- "" : un seul caractère (n'importe le quel)
- "%" : un nombre quelconque de caractères (y compris zéro caractères)
- le motif est insensible à la casse (ne distingue pas minuscules et majuscules)

# Clause WHERE : LIKE

```
SELECT titre, annee
FROM Film
WHERE titre LIKE '%ti%';
```



Films

| <i>titre</i> | <i>annee</i> | <i>realisateu</i> |
|--------------|--------------|-------------------|
| Alien        | 1979         | 1                 |
| Vertigo      | 1958         | 2                 |
| Psychose     | 1960         | 2                 |
| Kagemusha    | 1980         | 3                 |
| Volte-face   | 1997         | 4                 |
| Pulp Fiction | 1995         | 5                 |
| Titanic      | 1997         | 6                 |
| Sacrifice    | 1986         | 7                 |



|              |       |
|--------------|-------|
| titre        | annee |
| Pulp Fiction | 1995  |
| Vertigo      | 1958  |
| Titanic      | 1997  |

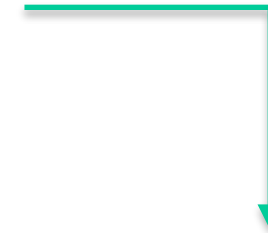
# Clause WHERE : LIKE

```
SELECT titre, annee  
FROM Film  
WHERE titre LIKE '_a%';
```



**Films**

| <i><b>titre</b></i> | <i><b>annee</b></i> | <i><b>realisateu</b></i> |
|---------------------|---------------------|--------------------------|
| Alien               | 1979                | 1                        |
| Vertigo             | 1958                | 2                        |
| Psychose            | 1960                | 2                        |
| Kagemusha           | 1980                | 3                        |
| Volte-face          | 1997                | 4                        |
| Pulp Fiction        | 1995                | 5                        |
| Titanic             | 1997                | 6                        |
| Sacrifice           | 1986                | 7                        |



|   |           |   |       |   |
|---|-----------|---|-------|---|
| + | -----     | + | ----- | + |
|   | titre     |   | annee |   |
| + | -----     | + | ----- | + |
|   | Kagemusha |   | 1995  |   |
|   | Sacrifice |   | 1958  |   |
| + | -----     | + | ----- | + |

# Clause WHERE : opérateurs logiques

On peut utiliser les opérateurs logiques **NOT**, **AND** et **OR** pour combiner différentes conditions

**NOT** condition

condition1 **AND** condition2

conditions1 **OR** condition2

# Clause WHERE : opérateurs logiques

SELECT titre, annee


FROM Films

WHERE titre BETWEEN 'Psychose' AND 'Titanic'

OR annee < 1965

Films

| <i><b>titre</b></i> | <i><b>annee</b></i> | <i><b>realisateu</b></i> |
|---------------------|---------------------|--------------------------|
| Alien               | 1979                | 1                        |
| Vertigo             | 1958                | 2                        |
| Psychose            | 1960                | 2                        |
| Kagemusha           | 1980                | 3                        |
| Volte-face          | 1997                | 4                        |
| Pulp Fiction        | 1995                | 5                        |
| Titanic             | 1997                | 6                        |
| Sacrifice           | 1986                | 7                        |



|              |       |
|--------------|-------|
| titre        | annee |
| Vertigo      | 1958  |
| Pulp Fiction | 1995  |
| Psychose     | 1960  |
| Titanic      | 1997  |
| Sacrifice    | 1986  |

# Clause WHERE : opérateurs logiques

Personne (id, prenom, nom, date\_naiss,...)

```
select nom, prenom
```

```
from Personne
```

```
where nom = 'Dupont'
```

```
and date_naiss > '2000-08-11';
```

```
select prenom, nom, date_naiss
```

```
from Personne
```

```
where ( prenom = 'Alice' and date_naiss <= '1980-12-24' )
```

```
or
```

```
( prenom <> 'Alice' and date_naiss is null ) ;
```

# Clause SELECT : DISTINCT

```
SELECT DISTINCT annee  
FROM Film
```

élimine les doublons  
dans le résultat

|       |
|-------|
| ----- |
| annee |
| ----- |
| 1979  |
| 1958  |
| 1960  |
|       |
| 1980  |
|       |
| 1997  |
|       |
| 1995  |
|       |
| 1986  |
| ----- |

Films

| titre        | annee | realisateu |
|--------------|-------|------------|
| Alien        | 1979  | 1          |
| Vertigo      | 1958  | 2          |
| Psychose     | 1960  | 2          |
| Kagemusha    | 1980  | 3          |
| Volte-face   | 1997  | 4          |
| Pulp Fiction | 1995  | 5          |
| Titanic      | 1997  | 6          |
| Sacrifice    | 1986  | 7          |



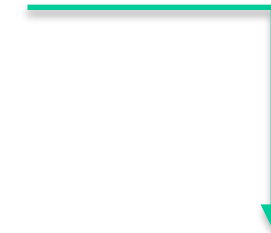
## Clause SELECT : renommage

```
SELECT titre AS titre_film  
FROM Films ;
```



**Films**

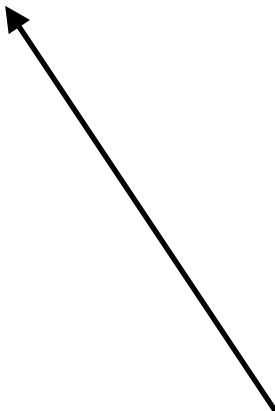
| <i><b>titre</b></i> | <i><b>annee</b></i> | <i><b>realisateu</b></i> |
|---------------------|---------------------|--------------------------|
| Alien               | 1979                | 1                        |
| Vertigo             | 1958                | 2                        |
| Psychose            | 1960                | 2                        |
| Kagemusha           | 1980                | 3                        |
| Volte-face          | 1997                | 4                        |
| Pulp Fiction        | 1995                | 5                        |
| Titanic             | 1997                | 6                        |
| Sacrifice           | 1986                | 7                        |



```
+-----+  
| titre_film |  
+-----+  
| Alien      |  
| Vertigo    |  
| Psychose   |  
| Kagemusha  |  
| Volte-face |  
| Pulp Fiction |  
| Titanic    |  
| Sacrifice  |  
+-----+
```

# Clause SELECT : attributs

```
SELECT Films.titre  
FROM Films
```



Optionnel (sauf en  
cas d'ambiguïté)

## Clause FROM : accéder à plusieurs tables

- Clause FROM : dans sa forme générale, admet une liste de plusieurs tables
- But : recomposer de l'information distribuée dans plusieurs tables
- Exemple : obtenir titre et réalisateur de tous les films

### Films

| <i>titre</i> | <i>annee</i> | <i>id_realisate</i> |
|--------------|--------------|---------------------|
| Alien        | 1979         | 1                   |
| Sacrifice    | 1986         | 6                   |

### Artistes

| <i>id</i> | <i>nom</i> | <i>prenom</i> | <i>naissan</i> |
|-----------|------------|---------------|----------------|
| 1         | Scott      | Ridley        | 1943           |
| 2         | Hitchcock  | Alfred        | 1899           |
| 3         | Kurosawa   | Akira         | 1910           |
| 4         | Woo        | John          | 1946           |
| 5         | Cameron    | James         | 1954           |
| 6         | Tarkovski  | Andrei        | 1932           |

## Clause FROM : accéder à plusieurs tables

- la clause FROM avec deux tables renvoie leur “produit” :
  - chaque ligne de la première table concaténée avec chaque ligne de la deuxième
  - appelé **produit cartésien**

```
SELECT * FROM Films, Artiste;
```

| titre     | année | lid_réalisateur | lid | nom       | prénom | naissance |
|-----------|-------|-----------------|-----|-----------|--------|-----------|
| Alien     | 1979  | 1               | 1   | Scott     | Ridley | 1943      |
| Alien     | 1979  | 1               | 2   | Hitchcock | Alfred | 1899      |
| Alien     | 1979  | 1               | 3   | Kurosawa  | Akira  | 1910      |
| Alien     | 1979  | 1               | 4   | Woo       | John   | 1946      |
| Alien     | 1979  | 1               | 5   | Tarkovski | Andrei | 1932      |
| Alien     | 1979  | 1               | 6   | Cameron   | James  | 1954      |
| Sacrifice | 1986  | 6               | 1   | Scott     | Ridley | 1943      |
| Sacrifice | 1986  | 6               | 2   | Hitchcock | Alfred | 1899      |
| Sacrifice | 1986  | 6               | 3   | Woo       | John   | 1946      |
| Sacrifice | 1986  | 6               | 4   | Kurosawa  | Akira  | 1910      |
| Sacrifice | 1986  | 6               | 5   | Cameron   | James  | 1954      |
| Sacrifice | 1986  | 6               | 6   | Tarkovski | Andrei | 1932      |

## Clause FROM : accéder à plusieurs tables

- La plupart du temps le produit cartésien contient des lignes “inutiles”
  - ▶ e.g. : pas significatif de concatener “Alien” avec “Hitchcock”
- On peut utiliser la condition WHERE pour sélectionner uniquement les lignes du produit qui sont “reliées”

```
SELECT * FROM Films, Artistes  
WHERE id_réalisateur = id ;
```

# Clause FROM : accéder à plusieurs tables

```
SELECT * FROM Films, Artistes
```



| titre     | année | lid_réalisateur | lid | nom       | prénom | naissance |
|-----------|-------|-----------------|-----|-----------|--------|-----------|
| Alien     | 1979  | 1               | 1   | Scott     | Ridley | 1943      |
| Alien     | 1979  | 1               | 2   | Hitchcock | Alfred | 1899      |
| Alien     | 1979  | 1               | 3   | Kurosawa  | Akira  | 1910      |
| Alien     | 1979  | 1               | 4   | Woo       | John   | 1946      |
| Alien     | 1979  | 1               | 5   | Tarkovski | Andrei | 1932      |
| Alien     | 1979  | 1               | 6   | Cameron   | James  | 1954      |
| Sacrifice | 1986  | 6               | 1   | Scott     | Ridley | 1943      |
| Sacrifice | 1986  | 6               | 2   | Hitchcock | Alfred | 1899      |
| Sacrifice | 1986  | 6               | 3   | Woo       | John   | 1946      |
| Sacrifice | 1986  | 6               | 4   | Kurosawa  | Akira  | 1910      |
| Sacrifice | 1986  | 6               | 5   | Cameron   | James  | 1954      |
| Sacrifice | 1986  | 6               | 6   | Tarkovski | Andrei | 1932      |

# Clause FROM : accéder à plusieurs tables

```
SELECT * FROM Films, Artistes  
WHERE id_réalisateur = id ;
```



| titre     | année | id_réalisateur | id | nom       | prénom | naissance |
|-----------|-------|----------------|----|-----------|--------|-----------|
| Alien     | 1979  | 1              | 1  | Scott     | Ridley | 1943      |
| Sacrifice | 1986  | 6              | 6  | Tarkovski | Andrei | 1932      |

## Clause FROM : accéder à plusieurs tables

```
SELECT * FROM Films, Artistes  
WHERE id_réalisateur = id ;
```

- Cela revient à concatener uniquement les lignes des deux tables qui satisfont la condition de WHERE

**Films**

| <i>titre</i> | <i>annee</i> | <i>id_realisate</i> |
|--------------|--------------|---------------------|
| Alien        | 1979         | 1                   |
| Sacrifice    | 1986         | 6                   |

**Artistes**

| <i>id</i> | <i>nom</i> | <i>prenom</i> | <i>naissan</i> |
|-----------|------------|---------------|----------------|
| 1         | Scott      | Ridley        | 1943           |
| 2         | Hitchcock  | Alfred        | 1899           |
| 3         | Kurosawa   | Akira         | 1910           |
| 4         | Woo        | John          | 1946           |
| 5         | Cameron    | James         | 1954           |
| 6         | Tarkovski  | Andrei        | 1932           |



# Clause FROM : accéder à plusieurs tables

```
SELECT * FROM Films, Artistes
WHERE id_réalisateur = id ;
```

| titre     | année | id_réalisateur | id | nom       | prénom | naissance |
|-----------|-------|----------------|----|-----------|--------|-----------|
| Alien     | 1979  | 1              | 1  | Scott     | Ridley | 1943      |
| Sacrifice | 1986  | 6              | 6  | Tarkovski | Andrei | 1932      |

- Ensuite la clause SELECT pour retenir uniquement les colonnes qui nous intéressent

```
SELECT titre, nom FROM Films, Artistes
WHERE réalisateur = id ;
```

| titre     | nom       |
|-----------|-----------|
| Alien     | Scott     |
| Sacrifice | Tarkovski |

## Clause FROM : Jointure

- L'opération de produit cartésien (`FROM Table1, Tables2`) suivie d'une condition de sélection (`WHERE <condition>`) est appelée **JOINTURE** (JOIN)
- `<condition>` est appelé **condition de jointure**
- Syntaxe alternative pour la même requête

```
SELECT titre, nom  
FROM Films JOIN Artistes ON (réalisateur = id );
```

- Des variantes de l'opérateur JOIN sont aussi disponibles (voir plus loin)

# Clause FROM : Jointure

- Un exemple plus général (chaque ligne est en jointure avec plusieurs lignes)

| X |   | Y |   |
|---|---|---|---|
| A | B | B | C |
| 1 | 2 | 1 | 6 |
| 4 | 7 | 2 | 2 |
| 4 | 2 | 4 | 2 |
| 8 | 3 | 4 | 3 |
| 5 | 4 |   |   |

```
SELECT  A, X.B, C
FROM    X, Y
WHERE   X.B = Y.B ;
```

| X,Y |     |     |     |
|-----|-----|-----|-----|
| X.A | X.B | Y.B | Y.C |
| 1   | 2   | 1   | 6   |
| 4   | 7   | 1   | 6   |
| 4   | 2   | 1   | 6   |
| 8   | 3   | 1   | 6   |
| 5   | 4   | 1   | 6   |
| 1   | 2   | 2   | 2   |
| 4   | 7   | 2   | 2   |
| 4   | 2   | 2   | 2   |
| 8   | 3   | 2   | 2   |
| 5   | 4   | 2   | 2   |
| 1   | 2   | 4   | 2   |
| 4   | 7   | 4   | 2   |
| 4   | 2   | 4   | 2   |
| 8   | 3   | 4   | 2   |
| 5   | 4   | 4   | 2   |
| 1   | 2   | 4   | 3   |
| 4   | 7   | 4   | 3   |
| 4   | 2   | 4   | 3   |
| 8   | 3   | 4   | 3   |
| 5   | 4   | 4   | 3   |

# Clause FROM : Jointure

- Un exemple plus général (chaque ligne est en jointure avec plusieurs lignes)

| X |   | Y |   |
|---|---|---|---|
| A | B | B | C |
| 1 | 2 | 1 | 6 |
| 4 | 7 | 2 | 2 |
| 4 | 2 | 4 | 2 |
| 8 | 3 | 4 | 3 |
| 5 | 4 |   |   |

```
SELECT  A, X.B, C
FROM X, Y
WHERE X.B = Y.B ;
```

X,Y

| X.A | X.B | Y.B | Y.C |
|-----|-----|-----|-----|
| 1   | 2   | 1   | 6   |
| 4   | 7   | 1   | 6   |
| 4   | 2   | 1   | 6   |
| 8   | 3   | 1   | 6   |
| 5   | 4   | 1   | 6   |
| 1   | 2   | 2   | 2   |
| 4   | 7   | 2   | 2   |
| 4   | 2   | 2   | 2   |
| 8   | 3   | 2   | 2   |
| 5   | 4   | 2   | 2   |
| 1   | 2   | 4   | 2   |
| 4   | 7   | 4   | 2   |
| 4   | 2   | 4   | 2   |
| 8   | 3   | 4   | 2   |
| 5   | 4   | 4   | 2   |
| 1   | 2   | 4   | 3   |
| 4   | 7   | 4   | 3   |
| 4   | 2   | 4   | 3   |
| 8   | 3   | 4   | 3   |
| 5   | 4   | 4   | 3   |

# Clause FROM : Jointure

| X |   | Y |   |
|---|---|---|---|
| A | B | B | C |
| 1 | 2 | 1 | 6 |
| 4 | 7 | 2 | 2 |
| 4 | 2 | 4 | 2 |
| 8 | 3 | 4 | 3 |
| 5 | 4 |   |   |

```
SELECT  A, X.B, C
FROM    X, Y
WHERE   X.B = Y.B ;
```

Résultat final :

| A | X.B | C |
|---|-----|---|
| 1 | 2   | 2 |
| 4 | 2   | 2 |
| 5 | 4   | 2 |
| 5 | 4   | 3 |

## Clause FROM : Jointure

- Un autre exemple : les noms et les notes des utilisateurs qui ont noté le film “Alien”

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

## Clause FROM : Jointure

- Un autre exemple : les noms et les notes des utilisateurs qui ont noté le film “Alien”

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

```
SELECT nom, note  
FROM Notation, Utilisateurs  
WHERE titre_film = 'Alien'  
AND Notation.pseudo = Utilisateurs.pseudo
```

### Alternative

```
SELECT nom, note  
FROM Notation JOIN Utilisateurs  
    ON (Notation.pseudo = Utilisateurs.pseudo)  
WHERE titre_film = 'Alien'
```

## Clause FROM : Jointure

- Un exemple plus complexe : les noms et les notes des utilisateurs qui ont noté les films de 1995

### Films

|       |       |                |
|-------|-------|----------------|
| titre | annee | id_realisateur |
|-------|-------|----------------|

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|



# Clause FROM : Jointure

- Un exemple plus complexe : les noms et les notes des utilisateurs qui ont noté les films de 1995

## Films

|       |       |                |
|-------|-------|----------------|
| titre | annee | id_realisateur |
|-------|-------|----------------|

## Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

## Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

```
SELECT nom, note  
FROM Notation, Utilisateurs, Films  
WHERE Notation.pseudo = Utilisateurs.pseudo  
AND titre_film = titre  
AND année = 1995
```

## Clause FROM : Jointure

- Un exemple plus complexe : les noms et les notes des utilisateurs qui ont noté les films de 1995

### Films

|       |       |                |
|-------|-------|----------------|
| titre | annee | id_realisateur |
|-------|-------|----------------|

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

Alternative :

```
SELECT nom, note
FROM Notation JOIN Utilisateurs ON (Notation.pseudo = Utilisateur.pseudo)
      JOIN Films ON (titre_film = titre)
WHERE année = 1995
```

## Clause FROM : Jointure

- Encore plus complexe : les noms et les notes des utilisateurs qui ont noté les films de 'Tarantino'

### Films

|       |       |                |
|-------|-------|----------------|
| titre | annee | id_realisateur |
|-------|-------|----------------|

### Artistes

|    |     |        |           |
|----|-----|--------|-----------|
| id | nom | prénom | naissance |
|----|-----|--------|-----------|

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

## Clause FROM : Jointure

- Encore plus complexe : les noms et les notes des utilisateurs qui ont noté les films de 'Tarantino'

### Films

|       |       |                |
|-------|-------|----------------|
| titre | annee | id_realisateur |
|-------|-------|----------------|

### Artistes

|    |     |        |           |
|----|-----|--------|-----------|
| id | nom | prénom | naissance |
|----|-----|--------|-----------|

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

```
SELECT Utilisateurs.nom, note
FROM Notation, Utilisateurs, Films, Artistes
WHERE Notation.pseudo = Utilisateurs.pseudo
AND titre_film = titre
AND id_realisateur = id
AND Artistes.nom = 'Tarantino'
```

## Clause FROM : Jointure

- Encore plus complexe : les noms et les notes des utilisateurs qui ont noté les films de 'Tarantino'

### Films

|       |       |                |
|-------|-------|----------------|
| titre | annee | id_realisateur |
|-------|-------|----------------|

### Artistes

|    |     |        |           |
|----|-----|--------|-----------|
| id | nom | prénom | naissance |
|----|-----|--------|-----------|

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

Alternative :

```
SELECT U.nom, note
FROM Notation JOIN Utilisateurs ON (Notation.pseudo = Utilisateur.pseudo)
      JOIN Films ON (titre_film = titre)
      JOIN Artistes ON (id_realisateur = id)
WHERE Artistes.nom = 'Tarantino'
```

# Clause FROM : Jointure naturelle

```
SELECT *  
FROM R NATURAL JOIN S
```

- Variante de l'opérateur JOIN
- exprime égalité des attributs avec le même nom (ON implicite)
- garde une seule instance de chaque attribut commun dans le résultat

## Clause FROM : Jointure naturelle

R

| A | B |
|---|---|
| 1 | 2 |
| 4 | 2 |
| 5 | 4 |

S

| B | C |
|---|---|
| 1 | 6 |
| 2 | 2 |
| 4 | 4 |

```
SELECT *  
FROM R NATURAL JOIN S
```



| A | B | C |
|---|---|---|
| 1 | 2 | 2 |
| 4 | 2 | 2 |
| 5 | 4 | 4 |

## Clause FROM : Jointure naturelle

R

| A | B |
|---|---|
| 1 | 2 |
| 4 | 2 |
| 5 | 4 |

S

| B | C |
|---|---|
| 1 | 6 |
| 2 | 2 |
| 4 | 4 |

```
SELECT *  
FROM R NATURAL JOIN S
```



- Equivalent à :

```
SELECT R.A, R.B, S.C  
FROM R JOIN S ON (  
    R.B = S.B  
)
```

| A | B | C |
|---|---|---|
| 1 | 2 | 2 |
| 4 | 2 | 2 |
| 5 | 4 | 4 |



## Clause FROM : Jointure naturelle

- Simplifie l'écriture de certaines requêtes, quand le schéma utilise les mêmes noms pour les clefs étrangères et les clefs référencées :
- Exemple : les noms et les notes des utilisateurs qui ont noté le film "Alien"

### Utilisateurs

|        |       |     |        |     |           |
|--------|-------|-----|--------|-----|-----------|
| pseudo | email | nom | prenom | mdp | code_pays |
|--------|-------|-----|--------|-----|-----------|

### Notation

|            |        |      |
|------------|--------|------|
| titre_film | pseudo | note |
|------------|--------|------|

```
SELECT nom, note  
FROM Notation NATURAL JOIN Utilisateurs  
WHERE titre_film = 'Alien'
```

## Clause FROM : renommage

```
SELECT F.titre  
FROM Films AS F
```

|              |
|--------------|
| AS optionnel |
|--------------|

Renommage des tables nécessaire si la même table est présente plusieurs fois dans la partie FROM, pour pouvoir distinguer les attributs

```
SELECT F1.titre  
FROM Film F1, Film F2  
WHERE F2.annee > F1.annee;
```

(les films qui ne sont pas le plus récents)

# Expressions

On peut utiliser des **expressions** dans les requêtes, à la place des attributs simples

Exemple : dans les clauses **SELECT, WHERE**

```
create table Article(  
  id integer primary key,  
  nom varchar(25) not null,  
  prix decimal(7,2),  
  prix_solde decimal(7,2),  
  check(prix_solde < prix)  
);
```

```
select nom, prix - prix_solde  
from Article  
where prix - prix_solde > 30 ;
```

## Article

| id | nom      | prix  | prix_solde |
|----|----------|-------|------------|
| 8  | pantalon | 99.99 | 59.95      |
| 5  | pull     | 40    | 26         |
| 3  | veste    | 120   | 79.95      |

| nom      | prix - prix_solde |
|----------|-------------------|
| pantalon | 40.04             |
| veste    | 40.05             |

# Fonctions et opérateurs prédéfinis

On peut utiliser des fonctions et opérateurs prédéfinis dans les expressions

Pour la plupart ce sont des ajouts du SGBD à la norme SQL

Exemples (PostgreSQL):

`ABS(num)` : valeur absolue

`str || str1` : concaténation des chaînes

`NOW()` : la date et heure courante

...

# Fonctions et opérateurs prédéfinis

```
SELECT 'réalisateur : ' || nom AS real
FROM Artistes ;
```



|         |                         |
|---------|-------------------------|
| +-----+ |                         |
|         | real                    |
| +-----+ |                         |
|         | réalisateur : Scott     |
|         | réalisateur : Hitchcock |
|         | réalisateur : Kurosawa  |
| +-----+ |                         |

```
SELECT EXTRACT(YEAR FROM NOW()) - naissance AS
age
FROM Artistes ;
```



|         |     |
|---------|-----|
| +-----+ |     |
|         | age |
| +-----+ |     |
|         | 73  |
|         | 117 |
|         | 106 |
| +-----+ |     |

Artistes

| id | nom       | prenom | naissan |
|----|-----------|--------|---------|
| 1  | Scott     | Ridley | 1943    |
| 2  | Hitchcock | Alfred | 1899    |
| 3  | Kurosawa  | Akira  | 1910    |

# Quelques fonctions et constantes PostgreSQL

`CEILING(num)`

Renvoie l'entier immédiatement supérieur ou égal à `num`

`FLOOR(num)`

Renvoie l'entier immédiatement inférieur ou égal à `num`

`current_date`

Renvoie la date courante

`current_time`

Renvoie l'heure courante

# Quelques fonctions et constantes PostgreSQL

```
CASE WHEN cond THEN val1  
      ELSE val2  
      END
```

Renvoie `val1` si `cond` est vrai, `val2` sinon (plusieurs `WHEN` possibles)

```
POSITION (substring IN string)
```

Position de `substring` dans `string`

```
LENGTH (str)
```

Renvoie la longueur de `str` (nombre de caractères)

# Trier les résultats des requêtes : ORDER BY

```
SELECT titre, annee
```

```
FROM Films
```

```
WHERE titre BETWEEN 'Psychose' AND 'Titanic'
```

```
ORDER BY titre;
```

| titre        | annee |
|--------------|-------|
| Psychose     | 1960  |
| Pulp Fiction | 1995  |
| Sacrifice    | 1986  |
| Titanic      | 1997  |



# Trier les résultats des requêtes : ORDER BY

On peut faire un tri sur plus d'une colonne

On peut trier dans l'ordre croissant (ASC) ou décroissant (DESC)

Exemple :

Liste les films par année et, dans une année, par ordre alphabétique inverse

```
SELECT annee, titre  
FROM Films  
ORDER BY annee ASC, titre DESC;
```

# Résultat

```
SELECT annee, titre FROM Films  
ORDER BY annee ASC, titre DESC;
```

| +-----+-----+ |              |
|---------------|--------------|
| annee         | titre        |
| +-----+-----+ |              |
| 1958          | Vertigo      |
| 1960          | Psychose     |
| 1979          | Alien        |
| 1980          | Kagemusha    |
| 1986          | Sacrifice    |
| 1995          | Pulp Fiction |
| 1997          | Volte-face   |
| 1997          | Titanic      |
| +-----+-----+ |              |

# Regroupement et agrégation en SQL

## Agréger des résultats : GROUP BY

Les agrégats SQL donnent la possibilité de combiner des valeurs sur plusieurs lignes d'une table

```
SELECT codePers, AVG(prix)
FROM Achat A, Produit PR
WHERE A.refProduit = PR.refProduit
GROUP BY codePers
HAVING AVG(prix) >= 100;
```

**Achat**

| codePers | refProduit |
|----------|------------|
| 123      | ma13       |
| 421      | pa23       |
| 567      | ma13       |
| 123      | ve2        |

**Produit**

| refProduit | Intitule | Prix |
|------------|----------|------|
| ma13       | manteau  | 100  |
| pa23       | pantalon | 85   |
| ju34       | jupe     | 63   |
| ve2        | veste    | 120  |

# GROUP BY : sémantique opérationnelle

```
SELECT codePers, AVG(prix)
```

```
FROM Achat A, Produit PR
```

```
WHERE A. refProduit = PR.refProduit
```

```
GROUP BY codePers
```

```
HAVING AVG(prix) >= 100;
```

I) La partie FROM-WHERE est évaluée:

| A.codePers | A.refProduit | P.refProduit | P.Intitule | P.Prix |
|------------|--------------|--------------|------------|--------|
| 123        | ma13         | ma13         | manteau    | 100    |
| 123        | ve2          | ve2          | veste      | 120    |
| 567        | ma13         | ma13         | manteau    | 100    |
| 421        | pa23         | pa23         | pantalon   | 85     |

# GROUP BY : sémantique opérationnelle

```
SELECT codePers, AVG(prix)
FROM Achat A, Produit PR
WHERE A. refProduit = PR.refProduit
GROUP BY codePers
HAVING AVG(prix) >= 100;
```

2) Le résultat est réparti en **groupes** de lignes. Dans chaque groupe : toutes les lignes avec une même valeur des attributs de GROUP BY:

| A.codePers | A.refProduit | P.refProduit | P.Intitule | P.Prix |
|------------|--------------|--------------|------------|--------|
| 123        | ma13         | ma13         | manteau    | 100    |
| 123        | ve2          | ve2          | veste      | 120    |
| 567        | ma13         | ma13         | manteau    | 100    |
| 421        | pa23         | pa23         | pantalon   | 85     |

## GROUP BY : sémantique opérationnelle

```
SELECT codePers, AVG(prix)
FROM Achat A, Produit PR
WHERE A. refProduit = PR.refProduit
GROUP BY codePers
HAVING AVG(prix) >= 100;
```

3) Les groupes qui satisfont la condition de HAVING restent :

| A.codePers | A.refProduit | P.refProduit | P.Intitule | P.Prix |
|------------|--------------|--------------|------------|--------|
| 123        | ma13         | ma13         | manteau    | 100    |
| 123        | ve2          | ve2          | veste      | 120    |
| 567        | ma13         | ma13         | manteau    | 100    |

# GROUP BY : sémantique opérationnelle

```
SELECT codePers, AVG(prix)
```

```
FROM Achat A, Produit PR
```

```
WHERE A. refProduit = PR.refProduit
```

```
GROUP BY codePers
```

```
HAVING AVG(prix) >= 100;
```

| codePers | AVG(prix) |
|----------|-----------|
| 123      | 110       |
| 567      | 100       |

4) SELECT est ensuite appliqué à chaque groupe

| A.codePers | A.refProduit | P.refProduit | P.Intitule | P.Prix |
|------------|--------------|--------------|------------|--------|
| 123        | ma13         | ma13         | manteau    | 100    |
| 123        | ve2          | ve2          | veste      | 120    |
| 567        | ma13         | ma13         | manteau    | 100    |



# GROUP BY : sémantique opérationnelle

```
SELECT codePers, AVG(prix), intitulé  
FROM Achat A, Produit PR  
WHERE A. refProduit = PR.refProduit  
GROUP BY codePers  
HAVING AVG(prix) >= 100;
```

| codePers | AVG(prix) |
|----------|-----------|
| 123      | 110       |
| 567      | 100       |

En présence d'agrégats, SELECT et HAVING font référence aux groupes et non pas aux lignes  $\Rightarrow$  peuvent mentionner :

- un attribut A, uniquement si A est dans le GROUP BY
- un agrégat F(A) sur n'importe quel attribut A

| A.codePers | A.refProduit | P.refProduit | P.Intitule | P.Prix |
|------------|--------------|--------------|------------|--------|
| 123        | ma13         | ma13         | manteau    | 100    |
| 123        | ve2          | ve2          | veste      | 120    |
| 567        | ma13         | ma13         | manteau    | 100    |

## Condition de HAVING

- Même syntaxe que la condition de WHERE ( conditions simples, opérateurs logiques, requêtes imbriquées, ...)
- Mais il faut en plus respecter la restriction citée plus haut :

( une condition HAVING peut mentionner :

- un attribut A, uniquement si A est dans le GROUP BY
- un agrégat F(A) sur n'importe quel attribut A

)

# Fonctions d'agrégation

```
SELECT codePers, AVG(prix)
FROM Achat A, Produit PR
WHERE A. refProduit = PR.refProduit
GROUP BY codePers
HAVING AVG(prix) >= 100;
```

D'autres fonctions d'agrégation:

SUM

COUNT - compte les lignes dans le groupe

MIN

MAX

etc.

# GROUP BY et HAVING optionnels

- En présence de fonctions d'agrégation,
  - ▶ le GROUP BY est optionnel
    - s'il manque, tout le résultat de la partie FROM-WHERE est considéré comme un seul groupe
  - ▶ HAVING est également optionnel
    - s'il manque, aucune condition n'est appliquée, et tous les groupes sont retenus

```
SELECT COUNT(refProduit)
FROM Produit
WHERE prix >= 100
```

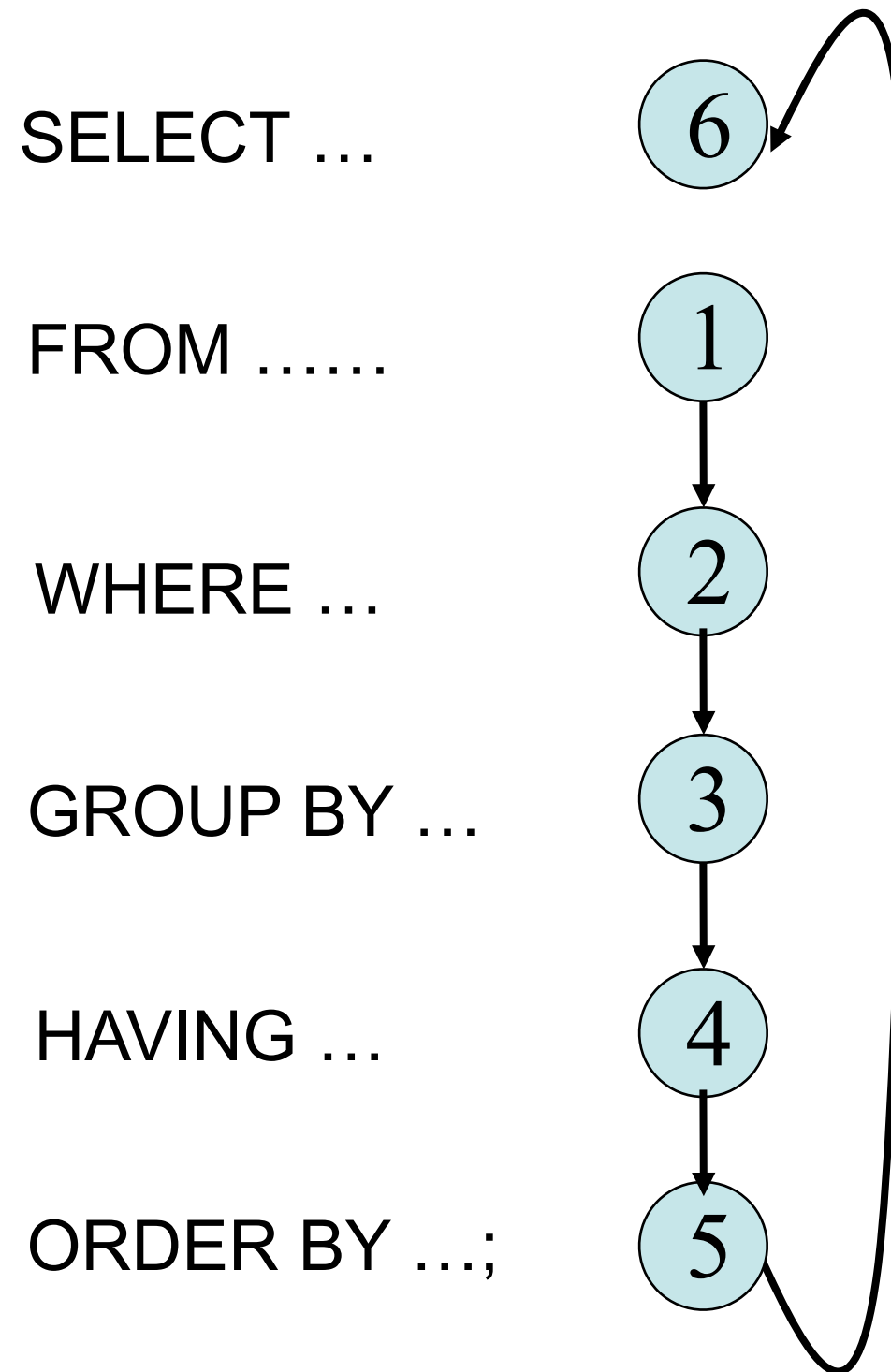
| COUNT(refProduit) |
|-------------------|
| 2                 |

```
SELECT codePers, COUNT(refProduit)
FROM Achat A
GROUP BY codePers
```

| codePers | COUNT(refProduit) |
|----------|-------------------|
| 123      | 2                 |
| 567      | 1                 |
| 421      | 1                 |

## Sémantique d'une requête

Pour comprendre la sémantique d'une requête on peut “imaginer” que les clauses sont exécutés dans cet ordre



# Agrégats et DISTINCT

article

| nom    | prix | qty |
|--------|------|-----|
| crayon | 1.40 | 20  |
| cahier | 2.10 | 15  |
| stylo  | 1.99 | 25  |
| gomme  | 0.90 | 15  |

```
select sum(qty) from article  
where qty <= 22;
```

20+15+15

```
select sum(distinct qty)  
where qty <=22;
```

20 + 15

```
select count(qty)  
from article;
```

→ 4

```
select count(distinct qty)  
from article;
```

→ 3

# Agrégats et expressions

L'argument d'une fonction d'agrégation peut être une expression quelconque sur les attributs disponibles :

article

| nom    | prix | qty |
|--------|------|-----|
| crayon | 1.40 | 20  |
| cahier | 2.10 | 15  |
| stylo  | 1.99 | 25  |
| gomme  | 0.90 | 15  |

```
select sum (prix*qty)
from article
where qty >= 20 ;
```

20\*1.40+25\*1.99

# Base de données pour les exemples : bibliothèque

```
CREATE TABLE lecteur(  
  id_lecteur int primary key,  
  nom varchar(30) not null,  
  prenom varchar(30),  
  date_naiss date,  
  sexe char(1));
```

```
CREATE TABLE livre(  
  id_livre int primary key,  
  titre varchar(40) not null);
```

```
CREATE TABLE auteur(  
  id_auteur int primary key,  
  nom varchar(30) not null,  
  prenom varchar(30),  
  date_naiss date);
```

```
CREATE TABLE livre_auteur(  
  id_livre int references livre,  
  id_auteur int references auteur,  
  primary key(id_livre,id_auteur));
```



# Base de données pour les exemples : bibliothèque

```
CREATE TABLE exemplaire(  
  ide int primary key,  
  id_livre int not null,  
  foreign key(id_livre) references livre);
```

```
CREATE TABLE emprunt(  
  date_empr date default current_date,  
  ide int references exemplaire,  
  id_lecteur int references lecteur,  
  date_prevue date not null,      – date prévue de retour  
  date_effective date default NULL, – date réelle de retour (NULL  
  si emprunt en cours)  
  primary key(date_empr,ide));
```

# Base de données bibliothèque - utilisation typique

Retour d'un livre à la bibliothèque (ide=12) (cf. plus loin pour les requêtes update):

```
update emprunt  
set date_effective = current_date  
where ide = 12;
```

Trouver les lecteurs qui ont au moins un emprunt en cours :

```
select nom, prénom  
from lecteur natural join emprunt  
where date_effective is null;
```

Un lecteur qui a emprunté plusieurs livres apparaîtra plusieurs fois.  
Pour éviter les doublons :

```
select DISTINCT nom, prénom  
from lecteur natural join emprunt  
where date_effective is null;
```

# Exemples de requêtes avec agrégats

Trouver le nombre de livres lus par chaque lecteur:

```
select nom, prenom, count(distinct id_livre) as nb_livres  
from lecteur natural join emprunt  
       natural join exemplaire  
group by id_lecteur, nom, prenom  
order by nb_livres desc;
```

Remarque : on compte les livres lus, pas les exemplaires

Remarque : les lecteurs avec 0 livres empruntés ne sont pas dans le résultat.

# Exemples de requêtes avec agrégats

Le nombre (d'exemplaires) de livres que détient chaque lecteur en ce moment :

```
select nom, prenom, count(*) as nb_livres  
from lecteur natural join emprunt  
where date_effective is null  
group by id_lecteur, nom, prenom  
order by nb_livres desc;
```

Remarque : les lecteurs sans emprunts en cours ne sont pas affichés.

## Exemples de requêtes avec agrégats

Les lecteurs qui détiennent au moins 4 exemplaires depuis au moins 1 mois.

Trier le résultat par nom et prénom.

```
select nom, prénom
from lecteur natural join emprunt
where date_effective is null
      and current_date - date_emprunt >= 30
group by id_lecteur, nom, prenom
having count(*) >= 4
order by nom, prenom;
```

# Manipulation de données

# Insérer des données

```
INSERT INTO Films
```

```
VALUES ('Pulp Fiction', 1995, 'Tarantino');
```

**Note** : on peut ne saisir que quelques attributs, et dans un ordre différent de celui défini pour la table.

```
INSERT INTO Films (realisateur, titre)
```

```
VALUES ('Allen', 'Match Point');
```

Si la valeur d'un attribut n'est pas spécifiée pendant l'insertion, la valeur NULL lui sera affectée

# Insérer des données

Resultat :

## Films

| <i><b>titre</b></i> | <i><b>annee</b></i> | <i><b>realisateur</b></i> |
|---------------------|---------------------|---------------------------|
| Alien               | 1979                | Scott                     |
| Vertigo             | 1958                | Hitchcock                 |
| Psychose            | 1960                | Hitchcock                 |
| Kagemusha           | 1980                | Kurosawa                  |
| Volte-face          | 1997                | Woo                       |
| Pulp Fiction        | 1995                | Tarantino                 |
| Titanic             | 1997                | Cameron                   |
| Sacrifice           | 1986                | Tarkovski                 |
| Match Point         | NULL                | Allen                     |

Attention : génère une erreur si les attributs pas spécifiés ont une contrainte NOT NULL



# Insertion de données et valeurs par défaut

Une clause **DEFAULT** peut être spécifiée pour un attribut :

```
CREATE TABLE Notation (  
    titre_film VARCHAR(50) NOT NULL,  
    pseudo VARCHAR(50) NOT NULL,  
    note INTEGER NOT NULL DEFAULT 0  
);
```

Si la valeur de l'attribut n'est pas spécifiée lors d'une insertion, sa valeur sera celle définie par la clause **DEFAULT**.

**INSERT INTO Notation VALUES ('Alien', 'jean87')** ne génère pas d'erreur et insère la ligne :

## Notation

| titre_film | pseudo | note |
|------------|--------|------|
| ...        | ...    | ...  |
| Alien      | jean87 | 0    |

## Insertion : clés à incrémentation automatique

- La plupart des SGBDs offrent la possibilité de définir des attributs dont la valeur est un entier automatiquement incrémenté à chaque insertion (pas présent dans le standard SQL)
- Très utile pour définir des identifiants “internes”

(syntaxe PostgreSQL)

```
CREATE TABLE Artiste (  
    id SERIAL PRIMARY KEY,  
    nom VARCHAR(50) NOT NULL,  
    prenom VARCHAR(50) NOT NULL,  
    naissance INTEGER,  
    UNIQUE (nom, prenom, naissance)  
);
```

- Pas besoin de fournir l'`id` lors de l'insertion d'un Artiste : il sera automatiquement affecté au `dernier id de la séquence + 1`

# Insertion : clés à incrémentation automatique

**Exemple.** Supposer la table *Artiste* initialement vide.

```
INSERT INTO Artiste (nom, prenom, naissance)  
VALUES ('Scott', 'Ridley', 1943);
```

```
INSERT INTO Artiste (nom, prenom, naissance)  
VALUES ('Hitchcock', 'Alfred', 1899);
```

- Remarque : on ne précise pas la clé "id".

## Insertion : clés à incrémentation automatique

```
SELECT * FROM Artiste;
```

| id | nom       | prenom | naissance |
|----|-----------|--------|-----------|
| 1  | Scott     | Ridley | 1943      |
| 2  | Hitchcock | Alfred | 1899      |

- La clé a été automatiquement générée (séquence croissante)

## Insertion : clés à incrémentation automatique

Une clé SERIAL peut également être précisée explicitement

```
INSERT INTO Artiste (id, nom, prenom, naissance)
```

```
VALUES (4, 'Woo', 'John', 1946);
```

```
SELECT * FROM Artiste;
```

| id | nom       | prenom | naissance |
|----|-----------|--------|-----------|
| 1  | Scott     | Ridley | 1943      |
| 2  | Hitchcock | Alfred | 1899      |
| 4  | Woo       | John   | 1946      |

## Insertion: reprise de l'incrémentation automatique

```
INSERT INTO Artiste (nom, prenom, naissance)
```

```
VALUES ('Kurosawa', 'Akira', 1910);
```

```
SELECT * FROM Artiste;
```

| ident | nom       | prenom | naissance |
|-------|-----------|--------|-----------|
| 1     | Scott     | Ridley | 1943      |
| 2     | Hitchcock | Alfred | 1899      |
| 4     | Woo       | John   | 1946      |
| 3     | Kurosawa  | Akira  | 1910      |

## Insertion : reprise de l'incrémentation automatique

| ident | nom       | prenom | naissance |
|-------|-----------|--------|-----------|
| 1     | Scott     | Ridley | 1943      |
| 2     | Hitchcock | Alfred | 1899      |
| 4     | Woo       | John   | 1946      |
| 3     | Kurosawa  | Akira  | 1910      |

```
INSERT INTO Artiste (nom, prenom, naissance)
VALUES ('Tarantino', 'Quentin', 1963);
```

```
ERROR:  duplicate key value violates unique constraint
DETAIL:  Key (id)=(4) already exists.
```

## Insertion : reprise de l'incrémentation automatique

La séquence continue toujours du dernier id généré, même si sa ligne a été supprimée, ou si l'insertion a échoué

```
DELETE FROM Artiste WHERE id > 2;
```

```
INSERT INTO Artiste (nom, prenom, naissance)
```

```
VALUES ('Kurosawa', 'Akira', 1910);
```

```
SELECT * FROM Artiste;
```

| ident | nom       | prenom | naissance |
|-------|-----------|--------|-----------|
| 1     | Scott     | Ridley | 1943      |
| 2     | Hitchcock | Alfred | 1899      |
| 5     | Kurosawa  | Akira  | 1910      |



# Clés à incrémentation automatique : reinitialisation

Pour réinitialiser le compteur de la séquence :

```
ALTER SEQUENCE Artiste_id_seq START WITH 4;
```



NomTable\_nomClef\_seq

# Clés à incrémentation automatique : reinitialisation

```
SELECT * FROM Artiste;
```

| id | nom       | prenom | naissance |
|----|-----------|--------|-----------|
| 1  | Scott     | Ridley | 1943      |
| 2  | Hitchcock | Alfred | 1899      |

```
ALTER SEQUENCE Artiste_id_seq START WITH 4;
```

```
INSERT INTO Artiste (nom, prenom, naissance)
```

```
VALUES ('Kurosawa', 'Akira', 1910);
```

```
SELECT * FROM Artiste;
```

| id | nom       | prenom | naissance |
|----|-----------|--------|-----------|
| 1  | Scott     | Ridley | 1943      |
| 2  | Hitchcock | Alfred | 1899      |
| 4  | Kurosawa  | Akira  | 1910      |

# Supprimer des données

```
SELECT * FROM Films
```

## Films

| <i>titre</i> | <i>annee</i> | <i>realisateur</i> |
|--------------|--------------|--------------------|
| Alien        | 1979         | Scott              |
| Vertigo      | 1958         | Hitchcock          |
| Psychose     | 1960         | Hitchcock          |
| Kagemusha    | 1980         | Kurosawa           |
| Volte-face   | 1997         | Woo                |
| Pulp Fiction | 1995         | Tarantino          |
| Titanic      | 1997         | Cameron            |
| Sacrifice    | 1986         | Tarkovski          |
| Match Point  | NULL         | Allen              |

```
DELETE FROM Films WHERE annee <= 1960;
```

## Supprimer des données

```
DELETE FROM Films WHERE annee <= 1960;
```

```
SELECT * FROM Films
```

### Films

| <i>titre</i> | <i>annee</i> | <i>realisateur</i> |
|--------------|--------------|--------------------|
| Alien        | 1979         | Scott              |
| Kagemusha    | 1980         | Kurosawa           |
| Volte-face   | 1997         | Woo                |
| Pulp Fiction | 1995         | Tarantino          |
| Titanic      | 1997         | Cameron            |
| Sacrifice    | 1986         | Tarkovski          |
| Match Point  | NULL         | Allen              |

Supprime toutes les lignes pour lesquelles l'année est inférieure ou égale à 1960.

WHERE est suivi d'une condition à évaluer sur chaque ligne

condition WHERE : même syntaxe que dans les requêtes SELECT

# Supprimer des données

```
DELETE FROM Films
```

Supprime toutes les lignes de la table Films  
(équivalent à `TRUNCATE Films`)

# Modifier des données

```
UPDATE Films
SET realisateur = 'Wu'
WHERE realisateur = 'Woo';
```

Met à jour le champs *réalisateur* de toutes les lignes sélectionnées par la clause *WHERE*.

## Films

| <i>titre</i> | <i>annee</i> | <i>realisateur</i> |
|--------------|--------------|--------------------|
| Alien        | 1979         | Scott              |
| Kagemusha    | 1980         | Kurosawa           |
| Volte-face   | 1997         | Wu                 |
| Pulp Fiction | 1995         | Tarantino          |
| Titanic      | 1997         | Cameron            |
| Sacrifice    | 1986         | Tarkovski          |
| Match Point  | NULL         | Allen              |

# Modifier des données

Forme generale :

```
UPDATE <table>
```

```
SET att1 = val1,..., attn = valn
```

```
WHERE <condition> ;
```

employe (id\_employe, nom, prenom, sexe, salaire, prime)

augmenter le salaire de 10% et donner une prime de 150 euros  
à toutes les femmes

```
update employe
```

```
set salaire = salaire * 1.1,
```

```
    prime = prime + 150
```

```
where sexe = 'F';
```

# Opérateurs ensemblistes en SQL



# UNION, INTERSECT, EXCEPT

```
(select ...) UNION [ ALL ] (select ...)
```

```
(select ...) INTERSECT [ALL] (select ...)
```

```
(select ...) EXCEPT [ ALL ] (select ...)
```

union, intersection, différence de deux tables, les doublons sont éliminés sauf si on ajoute ALL (avec ALL les doublons sont préservés).

Les deux tables doivent avoir le même nombre de colonnes, les types de colonnes correspondantes doit être identique.

# UNION, INTERSECT, EXCEPT

R

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |

S

| C | D | E |
|---|---|---|
| 3 | 8 | 6 |
| 3 | 8 | 7 |
| 1 | 2 | 8 |
| 2 | 5 | 8 |
| 2 | 8 | 9 |

(select \*  
from R)

union

(select c,d  
from S);

union ensembliste

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 2 | 5 |
| 2 | 8 |

# UNION, INTERSECT, EXCEPT

R

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |

S

| C | D | E |
|---|---|---|
| 3 | 8 | 6 |
| 3 | 8 | 7 |
| 1 | 2 | 8 |
| 2 | 5 | 8 |
| 2 | 8 | 9 |

(select \*  
from R) union all  
(select c,d  
from S);

pour chaque ligne dans l'union ensembliste,  
UNION ALL garde la somme des occurrences

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |
| 2 | 5 |
| 2 | 8 |

# UNION, INTERSECT, EXCEPT

R

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |

S

| C | D | E |
|---|---|---|
| 3 | 8 | 6 |
| 3 | 8 | 7 |
| 1 | 2 | 8 |
| 2 | 5 | 8 |
| 2 | 8 | 9 |

| A | B |
|---|---|
| 1 | 2 |
| 3 | 8 |

(select \*  
from R) intersect  
(select c,d  
from S);

intersection ensembliste

# UNION, INTERSECT, EXCEPT

R

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |

S

| C | D | E |
|---|---|---|
| 3 | 8 | 6 |
| 3 | 8 | 7 |
| 1 | 2 | 8 |
| 2 | 5 | 8 |
| 2 | 8 | 9 |

(select \*  
from R) intersect all  
(select c,d  
from S);

| A | B |
|---|---|
| 1 | 2 |
| 3 | 8 |
| 3 | 8 |

pour chaque ligne dans l'intersection ensembliste, INTERSECT ALL garde le nombre minimum d'occurrences

# UNION, INTERSECT, EXCEPT

R

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |

S

| C | D | E |
|---|---|---|
| 3 | 8 | 6 |
| 3 | 8 | 7 |
| 1 | 2 | 8 |
| 2 | 5 | 8 |
| 2 | 8 | 9 |

(select \*  
from R) except

(select c,d  
from S);

| A | B |
|---|---|
| 2 | 4 |
| 2 | 6 |

différence ensembliste

## UNION, INTERSECT, EXCEPT

R

| A | B |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |
| 3 | 8 |
| 3 | 8 |

S

| C | D | E |
|---|---|---|
| 3 | 8 | 6 |
| 3 | 8 | 7 |
| 1 | 2 | 8 |
| 2 | 5 | 8 |
| 2 | 8 | 9 |

(select \*  
from R) except all  
(select c,d  
from S);

| A | B |
|---|---|
| 2 | 4 |
| 2 | 6 |
| 3 | 8 |

Pour chaque ligne renvoyée par la requête de gauche, EXCEPT ALL garde le nombre d'occurrences à gauche moins le nombre d'occurrence à droite  
(aucune occurrence si ce nombre est négatif)

# Exemple de requête avec opérateurs ensemblistes

Trouver les identifiants et titres des livres lus exclusivement par les hommes :

```
(select id_livre, titre
from emprunt natural join exemplaire natural join livre
)
except
(select id_livre, titre
from lecteur natural join emprunt natural join exemplaire
        natural join livre
where sexe='F'
)
```



# Requêtes SQL imbriquées

# Une condition de WHERE plus complexe : IN

Les conditions de WHERE vues jusqu'à présent sont simples :

attribut op attribut ou attribut op valeur

Des conditions plus complexes sont possibles.

Rechercher des attributs appartenant à un ensemble :

```
SELECT titre FROM Films
```

```
WHERE nom IN ('Hitchcock', 'Scott', 'Kurosawa');
```

Equivalent à une suite de OR

```
SELECT titre FROM Films
```

```
WHERE nom = 'Hitchcock' OR nom = 'Scott' OR nom = 'Kurosawa';
```

## Une condition de WHERE plus complexe : IN

Plus intéressant : les valeurs de l'ensemble dans lequel rechercher peuvent être le résultat d'une (sous-) requête

```
SELECT id_realisateur FROM Films  
WHERE titre IN (SELECT titre FROM Notation WHERE note > 5);
```

Comme toute autre condition, la condition IN peut être combinée avec d'autres à l'aide des opérateurs logiques AND, OR, NOT

```
SELECT nom  
FROM Films, Artiste  
WHERE id = id_realisateur  
AND nom = 'Allen'  
AND titre NOT IN (SELECT titre FROM Notation);
```

# Conditions de WHERE /HAVING qui introduisent des sous-requêtes

IN

EXISTS

ALL, SOME, ANY



# IN

Forme générale (standard SQL)

```
(att1, ..., attk) IN (SELECT a1, ... ak, FROM ...)
```

ATTENTION: même arité

tous les SGBD ne l'implement pas dans cette forme

## Requêtes imbriquées

Les sous-requêtes peuvent contenir d'autres sous-requêtes (requêtes imbriquées)

Alternative pour la requête précédente (les livres qui n'ont jamais été empruntés)

```
select titre from livre
where id_livre NOT IN (select id_livre
                       from exemplaire
                       where id_exemplaire IN
                           (select id_exemplaire
                            from emprunt) );
```

**MOIN EFFICACE !**

En général :

plus le niveau d'imbrication est élevé, moins efficace est l'exécution de la requête

# EXISTS

EXISTS (SELECT ...)

est vrai si SELECT retourne une table non vide.

Trouver les lecteurs qui ont effectué des emprunts :

```
select nom, prénom  
from lecteur L  
where exists(  
    select *  
    from emprunt E  
    where E.id_lecteur = L.id_lecteur  
);
```



# EXISTS et sous-requêtes corrélées

Une sous-requête (introduite par n'importe quelle condition) est en principe évaluée une fois pour chaque ligne de la requête principale :

```
select nom, prénom
```

```
from lecteur L
```

```
where exists(
```

```
    select *
```

```
    from emprunt E
```

```
    where E.id_lecteur = L.id_lecteur
```

```
);
```

évaluée une fois pour chaque  
lecteur

⇒ la sous-requête peut faire référence à la ligne courante de la requête principale (le lecteur courant dans l'exemple)

Ce genre de sous-requête est appelé **sous-requête corrélée**

Particulièrement utile avec la condition EXISTS, mais possible avec tous les autres types de sous-requête

# EXISTS

EXISTS est surtout utilisé avec la négation

Trouver les lecteurs qui n'ont jamais effectué d'emprunt :

```
select nom, prénom  
from lecteur L  
where not exists(  
    select *  
    from emprunt E  
    where E.id_lecteur = L.id_lecteur  
);
```

# EXISTS

EXISTS est surtout utilisé avec la négation

En effet EXISTS sans négation peut être simulé par un jointure (plus efficace)

```
select nom, prénom  
from lecteur L  
where exists(  
    select *  
    from emprunt E  
    where E.id_lecteur  
        = L.id_lecteur  
);
```

équivalent  
à

```
select distinct nom, prénom  
from lecteur L, emprunt E  
where E.id_lecteur  
    = L.id_lecteur
```

# NOT IN, NOT EXISTS et EXCEPT

Les requêtes qui nécessitent une forme de négation peuvent en général être exprimées à la fois avec EXCEPT, NOT IN et NOT EXISTS

Mais les trois formulations sont différentes

Trouver les lecteurs qui n'ont jamais effectué d'emprunt :

```
select nom, prénom
from lecteur L
where NOT EXISTS(
    select *
    from emprunt E
    where E.id_lecteur = L.id_lecteur
);
```

# NOT IN, NOT EXISTS et EXCEPT

Les requêtes qui nécessitent une forme de négation peuvent en général être exprimées à la fois avec EXCEPT, NOT IN et NOT EXISTS

Mais les trois formulations sont différentes

Trouver les lecteurs qui n'ont jamais effectué d'emprunt :

```
select nom, prénom  
from lecteur L  
where id_lecteur NOT IN (  
    select id_lecteur  
    from emprunt  
);
```

# NOT IN, NOT EXISTS et EXCEPT

Les requêtes qui nécessitent une forme de négation peuvent en général être exprimées à la fois avec EXCEPT, NOT IN et NOT EXISTS

Mais les trois formulations sont différentes

Trouver les lecteurs qui n'ont jamais effectué d'emprunt :

```
select nom, prénom  
from lecteur where id_lecteur IN  
(select id_lecteur  
from lecteur  
EXCEPT  
select id_lecteur  
from emprunt)
```

# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui n'ont emprunté aucun livre d'Alexandre Dumas :

```
select nom, prenom
from lecteur L
where NOT EXISTS (
    select id_exemplaire
    from emprunt natural join
        exemplaire natural join
            livre natural join livre_auteur
            natural join auteur
    where nom = 'Dumas' and
        prenom = 'Alexandre'
    and id_lecteur = L.id_lecteur
);
```

tous les id\_exemplaire des livres  
de Dumas  
empruntés par le lecteur L

# Requêtes typiques qui nécessitent une ou plusieurs négations

trouver les livres qui se trouvent dans le catalogue de livres  
et dont la bibliothèque ne possède aucun exemplaire :

```
select titre  
from livre L  
where NOT exists (  
    select * from exemplaire E  
    where E.id_livre = L.id_livre  
);
```



# Requêtes typiques qui nécessitent une ou plusieurs négations

Parfois la négation est explicite dans la formulation:

Exemple : Les lecteurs qui **n'ont pas** d'emprunts en cours

D'autres fois la négation est cachée :

Exemple :

Les livres qui ont été empruntés uniquement par des hommes

⇔

Les livres qui **n'ont jamais** été empruntés par une femme

# Requêtes typiques qui nécessitent une ou plusieurs négations

Les livres qui ont été empruntés uniquement par des hommes

```
(select id_livre, titre  
from emprunt natural join exemplaire natural join livre  
)
```

EXCEPT

```
(select id_livre, titre  
from lecteur natural join emprunt natural join exemplaire  
natural join livre  
where sexe='F'  
)
```

# Requêtes typiques qui nécessitent une ou plusieurs négations

Les livres qui ont été empruntés uniquement par des hommes

```
select id_livre, titre
from livre
where id_livre NOT IN (
    select id_livre
    from lecteur natural join emprunt natural join exemplaire
    where sexe='F'
)
```

# Requêtes typiques qui nécessitent une ou plusieurs négations

Les livres qui ont été empruntés uniquement par des hommes

```
select id_livre, titre
from livre L
where NOT EXISTS (
    select *
    from lecteur natural join emprunt natural join exemplaire
    where id_livre = L.id_livre
    and sexe='F'
)
```

# Requêtes typiques qui nécessitent une ou plusieurs négations

Raisons de la négation “cachée” :

- SQL n’a pas d’opérateurs explicites pour exprimer qu’une propriété est vérifiée **pour tous** les éléments d’un ensemble  
(Ex. le lecteur dans **tous les emprunts** d’une livre est une homme)
- SQL fournit un moyen explicite uniquement pour exprimer qu’il **existe** un élément d’un ensemble qui satisfait une propriété  
(Ex. **il existe** un emprunt dont le lecteur est une femme)
- Mais la combinaison de **négation** et “**il existe**” peut exprimer “**pour tous**” !  
(Ex.  
il **n’existe pas** un emprunt dont le lecteur est une femme  
 $\Leftrightarrow$  dans **tous les emprunts** le lecteur est un homme)

# Requêtes typiques qui nécessitent une ou plusieurs négations

Parfois plusieurs négations (imbriquées) sont “cachées”

C'est le cas pour toutes les requêtes qui demandent de vérifier  
l'inclusion ou l'égalité de deux ensembles

$$A \subseteq B \quad \text{ou} \quad A = B$$

Exemples :

- Trouver les étudiants qui ont pris tous les cours obligatoires
  - ▶ (vérifier :  $\text{cours obligatoires} \subseteq \text{cours pris par l'étudiant}$  )
- Trouver les lecteurs qui ont lu tous les livres de Alexandre Dumas
  - ▶ (vérifier :  $\text{livres de Alexandre Dumas} \subseteq \text{livres lus par le lecteur}$  )
- Trouver les lecteurs qui ont lu exactement les livres d'Alexandre Dumas ( c'est à dire tous les livres de Dumas et et aucun autre livre).
  - ▶ (vérifier :  $\text{livres de Alexandre Dumas} = \text{livres lus par le lecteur}$  )

# Requêtes typiques qui nécessitent une ou plusieurs négations

Re-formulation de inclusion et égalité d'ensembles avec deux négations imbriquées

$$A \subseteq B$$

équivalent à la condition

NOT EXISTS (A EXCEPT B)

$$A = B$$

équivalent à la condition

(A  $\subseteq$  B) AND (B  $\subseteq$  A)

# Requêtes typiques qui nécessitent une ou plusieurs négations

Re-formulation de inclusion et égalité d'ensembles avec deux négations imbriquées

Plus en général :  
chacune des deux  
négations peut être  
exprimée par  
NOT EXISTS ou NOT IN  
ou EXCEPT

$$A \subseteq B$$

équivalent à la condition

NOT EXISTS (A EXCEPT B)

$$A = B$$

équivalent à la condition

(A  $\subseteq$  B) AND (B  $\subseteq$  A)



# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui ont lu tous les livres d'Alexandre Dumas :

```
select nom, prenom from lecteur L
where NOT EXISTS (
  (select id_livre
   from livre_auteur natural join auteur
   where nom='Dumas' and prenom='Alexandre')
  EXCEPT
  (select id_livre
   from exemplaire natural join emprunt
   where id_lecteur = L.id_lecteur)
)
```

**rouge** - l'ensemble des livres d'Alexandre Dumas (A)

**vert** - l'ensemble des livres lus par le lecteur L (B)

# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui ont lu tous les livres d'Alexandre Dumas :

```
select nom, prenom from lecteur L
where NOT EXISTS (
    select id_livre
    from livre_auteur natural join auteur
    where nom='Dumas' and prenom='Alexandre'
    and id_livre NOT IN (
        select id_livre
        from exemplaire natural join emprunt
        where id_lecteur = L.id_lecteur
    )
)
```

**rouge** - l'ensemble des livres d'Alexandre Dumas

**vert** - l'ensemble des livres lus par le lecteur L

# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui ont lu tous les livres d'Alexandre Dumas :

```
select nom, prenom from lecteur L
where NOT EXISTS (
    select id_livre
    from livre_auteur natural join auteur
    where nom='Dumas' and prenom='Alexandre'
    and (id_livre, L.id_lecteur) NOT IN (
        select id_livre, id_lecteur
        from exemplaire natural join emprunt
    )
)
```

**rouge** - l'ensemble des livres d'Alexandre Dumas

**vert** - les livres avec leurs lecteurs

# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui ont lu tous les livres d'Alexandre Dumas :

```
select nom, prenom from lecteur L
where NOT EXISTS (
  select id_livre
  from livre_auteur LA natural join auteur
  where nom='Dumas' and prenom='Alexandre'
  and NOT EXISTS (
    select *
    from exemplaire natural join emprunt
    where id_livre = LA.id_livre
    and id_lecteur = L.id_lecteur
  )
)
```

beaucoup  
de corrélation !

rouge - l'ensemble des livres d'Alexandre Dumas

vert - les emprunts du livre LA.id\_livre par le lecteur L.id\_lecteur

# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui ont lu tous les livres d'Alexandre Dumas :

```
select nom, prenom from lecteur
where id_lecteur NOT IN (
    select id_lecteur
    from lecteur, livre_auteur natural join auteur
    where nom='Dumas' and prenom='Alexandre'
    and (id_livre, id_lecteur) NOT IN (
        select id_livre, id_lecteur
        from exemplaire natural join emprunt
    )
)
```

moins “naturelle”

**rouge** - les lecteurs pour les quels il existe un livre de Dumas

**vert** - les livres avec leurs lecteurs

# Requêtes typiques qui nécessitent une ou plusieurs négations

Trouver les lecteurs qui ont lu tous les livres d'Alexandre Dumas :

```
select nom, prenom from lecteur
where id_lecteur NOT IN (
  select id_lecteur
  from lecteur, livre_auteur natural join auteur
  where nom='Dumas' and prenom='Alexandre'
  and (id_livre, id_auteur) in (
    select id_livre, id_auteur
    from livre_auteur natural join emprunt
    where nom='Dumas' and prenom='Alexandre'
  )
)
```

Encore d'autres façons d'écrire ce genre de requête plus loin  
(en utilisant les agrégats)

moins “naturelle”

rouge - les lecteurs pour les quels il existe un livre de Dumas

vert - les livres avec leurs lecteurs

# ALL, SOME, ANY

attribut op ANY (SELECT b FROM ...)

SOME  
synonyme  
de ANY

attribut op ALL (SELECT b FROM ...)

op:=        =        <>        <=        >=        >        <

A > ANY (select ...)

équivalent à

A>a1 OR A>a2 OR ... OR A>an

où a1,a2,...,an est le résultat de select

# ALL, SOME, ANY

attribut op ANY (SELECT b FROM ...)

SOME  
synonyme  
de ANY

attribut op ALL (SELECT b FROM ...)

op:=        =        <>        <=        >=        >        <

A > ALL (select ...)

équivalent à

A>a1 AND A>a2 AND ... AND A>an

où a1,a2,...,an est le résultat de select



# ALL, SOME, ANY

BD bibliothèque :

calculer le nombre d'exemplaires de livres d'Alexandre Dumas

```
select count(id_exemplaire)
from exemplaire
where id_livre = ANY (
    select id_livre
    from livre_auteur natural join auteur
    where nom='Dumas' and prenom='Alexandre'
);
```

Remarque :

= **ANY** est équivalent à **IN**

# ALL, SOME, ANY

employe (nom, prénom, sexe, salaire,departement)

Trouver les employés qui gagnent plus que tous les employés femmes :

```
select nom, prénom
from employe
where salaire > ALL (
    select salaire
    from employe
    where sexe='F'
);
```

## ALL, SOME, ANY

Trouver les employés dont le salaire est supérieur aux salaires moyens de tous les départements :

```
select nom, prénom
from employe where salaire > ALL (
    select avg(salaire)
    from employe
    group by departement );
```

Trouver les départements avec le salaire moyen le plus élevé :

```
select département
from employe
group by department
having avg(salaire) >= ALL(
    select avg(salaire)
    from employe
    group by department);
```

# ALL, SOME, ANY

Calculer, pour chaque département, les employés qui ont le salaire le plus élevé dans ce département

Première version :

```
select département, nom, prénom  
from employe E where salaire >= ALL (  
    select salaire  
    from employe  
    where department = E.département);
```

# ALL, SOME, ANY

Calculer, pour chaque département, les employés qui ont le salaire le plus élevé dans ce département

Deuxième version :

```
select département, nom, prénom  
from employe E where salaire = ANY (  
    select MAX(salaire)  
    from employe  
    where department = E.département);
```

# ALL, SOME, ANY

Calculer, pour chaque département, les employés qui ont le salaire le plus élevé dans ce département

Troisième version (évite la sous-requête corrélée) :

```
select département, nom, prénom  
from employe  
where (département, salaire) = ANY (  
    select département, MAX(salaire)  
    from employe  
    group by département  
);
```

← les requêtes de ANY et ALL  
peuvent avoir arité > 1  
(tout comme IN)

# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

```
select département, nom, prénom  
from employe E where salaire = ANY (  
    select MAX(salaire)  
    from employe  
    where  department = E.département);
```

# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

```
select département, nom, prénom  
from employe E where salaire = (  
    select MAX(salaire)  
    from employe  
    where  department = E.département);
```



# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

Trouver les employés dont le salaire est supérieur au salaire moyen

```
select nom, prénom  
from employe where salaire > (  
    select avg(salaire)  
    from employe  
);
```

# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

Trouver les employés dont le salaire est supérieur au salaire moyen de leur département

```
select nom, prénom  
from employe E  
where salaire > (select avg(salaire)  
                  from employe  
                  departement = E.departement);
```

# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

Trouver les département dont le salaire moyen est plus élevé que le salaire moyen de l'entreprise :

```
select département
from employe
group by department
having avg(salaire) > (select avg(salaire)
                        from employe);
```

# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

trouver les salariés qui gagnent plus que le 75% du salaire moyen :

```
select nom, prénom  
from employe  
where salaire > 0.75*(select avg(salaire) from employe);
```

# ALL, SOME, ANY

Quand la sous-requête introduite par ANY renvoie toujours une seule ligne, ANY peut être omis :

trouver les salariés qui gagnent entre 75% et 125% du salaire moyen :

```
select nom, prénom  
from employe  
where salaire between 0.75*(select avg(salaire) from employe) and  
1.25*(select avg(salaire) from employe) ;
```

## Sous-requêtes qui retournent une seule valeur

Par extension, une sous-requête qui retourne une seule valeur peut être utilisée partout où on s'attend une valeur

trouver les lecteurs qui ont emprunté plus en 2015 qu'en 2014 :

```
select nom, prenom
from lecteur L
where (select count(*)
      from emprunt E
      where extract(year from date_emprunt)=2015
      and E.id_lecteur=L.id_lecteur) >
      (select count(*)
      from emprunt E
      where extract(year from date_emprunt)=2014
      and E.id_lecteur=L.id_lecteur) ;
```

# Inclusion et égalité d'ensembles avec les agrégats

$A \subseteq B$  et  $A = B$  peuvent être exprimés sans négation si on utilise les agrégats

$$A \subseteq B$$

équivalent à la condition

$$|A \cap B| = |A|$$

(nombre d'éléments à la fois dans A et dans B = nombre d'éléments dans A)

## Exemple

Les lecteurs qui ont lu tous les livres d'Alexandre Dumas sont  
les lecteurs pour qui :

nombre de livres d'Alexandre Dumas lus par le lecteur ( $|A \cap B|$ ) =  
nombre de livres d'Alexandre Dumas ( $|A|$ )

# Inclusion et égalité d'ensembles avec les agrégats

Les lecteurs qui ont lu tous les livres d'Alexandre Dumas

```
select L.nom, L.prenom
from lecteur L JOIN (emprunt NATURAL JOIN exemplaire
    NATURAL JOIN livre_auteur NATURAL JOIN auteur) J
    ON ( L.id_lecteur = J.id_lecteur)
where J.nom='Dumas' and J.prenom='Alexandre'
group by L.id_lecteur, L.nom, L.prenom
having count (distinct id_livre) = (
    select count (*)
    from livre_auteur natural join auteur
    where nom='Dumas' and prenom='Alexandre'
)
```



# Inclusion et égalité d'ensembles avec les agrégats

$A \subseteq B$  et  $A = B$  peuvent être exprimés sans negation si on utilise les agrégats

$$A = B$$

équivalent à la condition

$$|A \cap B| = |A| = |B|$$

(nombre d'elements à la fois dans A et dans B = nombre d'elements dans A = nombre d'elements dans B )

## Exemple

Les lecteurs qui ont lu tous les livres d'Alexandre Dumas et aucun autre livre sont les lecteurs pour qui :

nombre de livres d'Alexandre Dumas lus par le lecteur ( $|A \cap B|$ ) =

nombre de livres d'Alexandre Dumas ( $|A|$ ) =

nombre de livres lus par le lecteur ( $|B|$ )

# Inclusion et égalité d'ensembles avec les agrégats

Les lecteurs qui ont lu tous les livres d'Alexandre Dumas et aucun autre livre

```
select L.nom, L.prenom
from lecteur L JOIN (emprunt NATURAL JOIN exemplaire
    NATURAL JOIN livre_auteur NATURAL JOIN auteur) J
    ON ( L.id_lecteur = J.id_lecteur)
where J.nom='Dumas' and J.prenom='Alexandre'
group by L.id_lecteur, L.nom, L.prenom
having count (distinct id_livre) = (
    select count (*)
    from livre_auteur natural join auteur
    where nom='Dumas' and prenom='Alexandre')
and count (distinct id_livre) = (
    select count (distinct id_livre)
    from emprunt NATURAL JOIN exemplaire
    where id_lecteur = L.id_lecteur)
```