BDW1 : Bases de données et Programmation Web

SQL : Partie 2 - Sous-requêtes et regroupements

Nicolas Lumineau

nicolas.lumineau@univ-lyon1.fr

Université Claude Bernard Lyon 1

Licence 2^e année - 2016/2017

Enseignements de BDW1

1. [BD & Web] Introduction
2. [BD]La syntaxe SQL
   - SQL : de la projection à la jointure
   - SQL : sous-requêtes et regroupements
   - Fonctions diverses et commandes SQL
3. [BD]Conception de base de données
   - Schéma Entité/Association
   - Du modèle conceptuel au modèle relationnel
4. [Web] Programmation Web
   - Le langage HTML/CSS
   - Le langage PHP
     - Interrogation d'une BD via PHP
5. [BD] Optimisation de requête
   - Algèbre Relationnelle
   - Transformation d'arbres algébriques

Plan

Remarque

Toutes les requêtes que nous avons vues jusqu'à présent se suffisaient pour répondre au besoin exprimé par l'utilisateur.

Exemple

- Donner le prénom et le prénom des artistes qui ont joué dans les mêmes films que 'Tom CRUISE'.

Pour construire la requête de l'exemple, il est possible de raisonner de la manière suivante :

- Commençons par identifier les identifiants des films dans lesquels l'acteur 'Tom CRUISE' a joué
- Une fois ces films identifiés, regardons si un(e) artiste a un rôle dans l'un de ces films
- Retournons le prénom et le nom de ces artistes

Vers la définition de sous-requêtes...

Exemple

- Donner le prénom et le prénom des artistes qui ont joué dans les mêmes films que 'Tom CRUISE'.

Pour raisonner de cette manière, nous sommes partis du principe qu'il était possible de calculer les identifiants des films de 'Tom CRUISE' à l'intérieur de la requête.

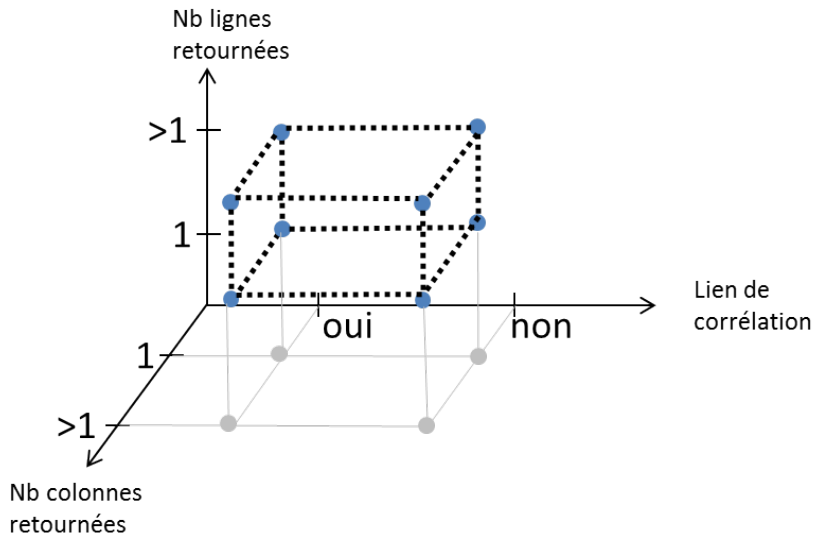
Ceci est possible car nous pouvons créer des **sous-requêtes** dans une requête.
$\Rightarrow$ Augmentation de l'expression du langage de requête.

Hiérarchisation des requêtes

Considérer des sous-requêtes a pour conséquence de créer différents niveaux de requêtes. Nous distinguons alors :

- la requête principale
- la sous-requête,
  qui retourne :
  - un tuple ou un ensemble de tuples
  - un attribut ou un ensemble d'attributs

  et qui peut être :
  - non corrélée à la requête principale
  - corrélée à la requête principale



Remarque:

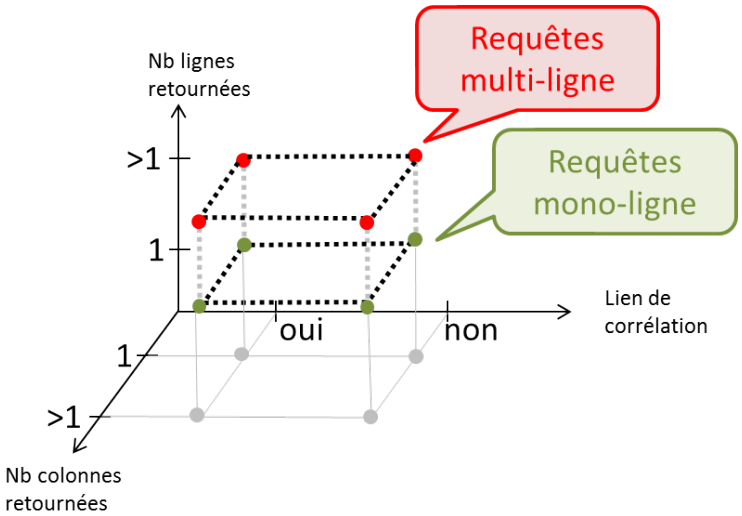
- la requête principale est celle associée au premier SELECT
- une sous-requête peut contenir elle-même une autre sous-requête ⇒ concept d'imbrication des requêtes !

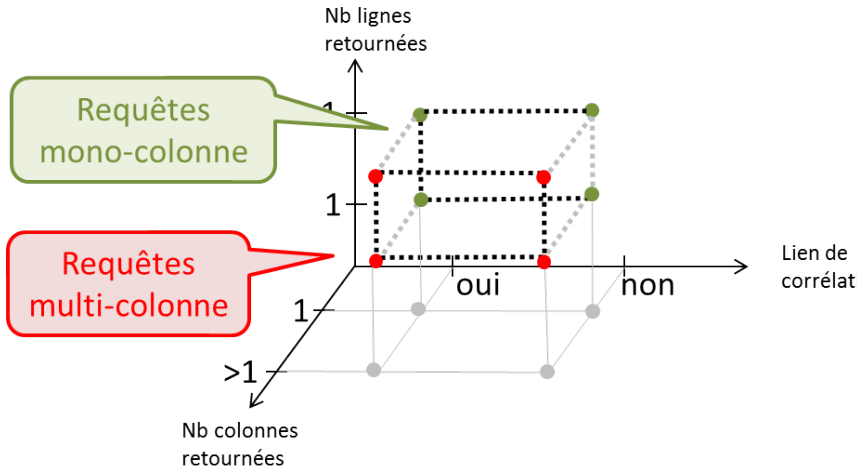
Utiliser des sous-requêtes : mais où ?

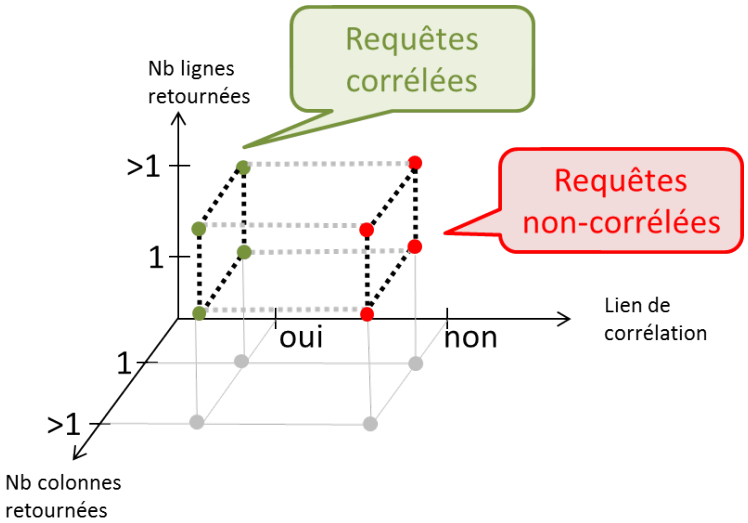
- Dans un premier temps, nous allons voir des sous-requêtes dans les clauses :
  - **WHERE**
  - **FROM** (à condition de renommer le résultat)

  Plus tard, nous verrons que les sous-requêtes sont également utilisable dans les clauses :
  - **SELECT** (à condition que pour chaque ligne sélectionnée par la requête principale, on ne sélectionne qu'une ligne dans la sous-requête composé d'un seul attribut)
  - **HAVING** (pour des conditions portant sur des groupes)

Différents types de sous-requête

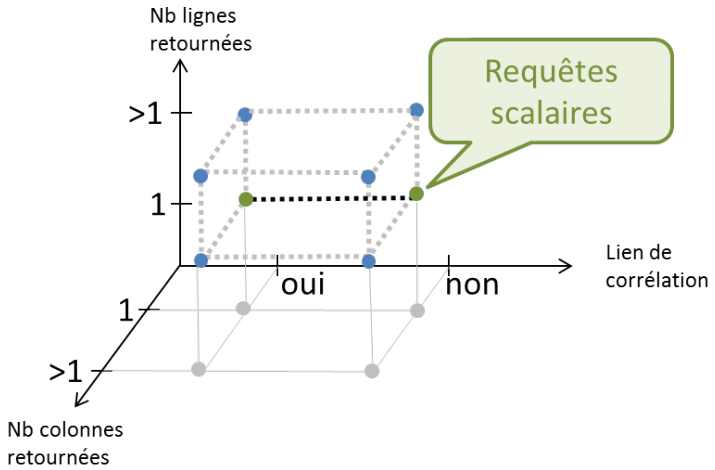






Les sous-requêtes scalaires

Une sous-requête qui ne retourne qu'un seul tuple et qu'un seul attribut est nommé : **Sous-requête scalaire**.



Comme la sous-requête ne retourne qu'une valeur d'un seul attribut, il est alors possible de comparer ($=$, $!=$, $<$, $>$, ...) directement le résultat de requête avec la valeur d'un attribut de la requête principale.

Cas où la sous-requête scalaire avec l'égalité comme comparaison

SELECT *
FROM $nomTable_1$
WHERE $att_1 =$ (**SELECT** att_3 **FROM** $nomTable_2$) ;

Exemple 1 de sous-requête

Donner le prénom et le nom des artistes qui ont déjà gagné un salaire supérieur au sens strict au salarie gagné pour le rôle de "George Valentin" dans le film "The Artist".

```
SELECT A.prenom, A.nom
 FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
 WHERE JD.sal > (SELECT JD.sal
                 FROM Film F JOIN JoueDans JD
                 ON F.idf = JD.idf
                 WHERE F.titre = 'The Artist'
                 AND JD.nom = 'George Valentin');
```

Résultat :

prenom	nom
Scarlett	JOHANSSON
Tom	CRUISE
Dany	BOON

Exemple 1 de sous-requête

En effet : les salaires supérieurs à "0.3" sont "1.3", "14", "20".

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

JoueDans(ida, idf, nom)

idp	idf	nom	sal
1	3	George Valentin	0,3
1	4	Hubert Bonisseur de La Bath	0,2
2	2	Lucy Miller	14
3	1	Bill Cage	20
4	4	Larmina El Akmar Betouche	NULL
4	3	Peppy Miller	NULL
5	5	Antoine Bailleul	1,3
6	1	Rita Vrataski	NULL

Film(idf, titre, annee, genre, real, budget)

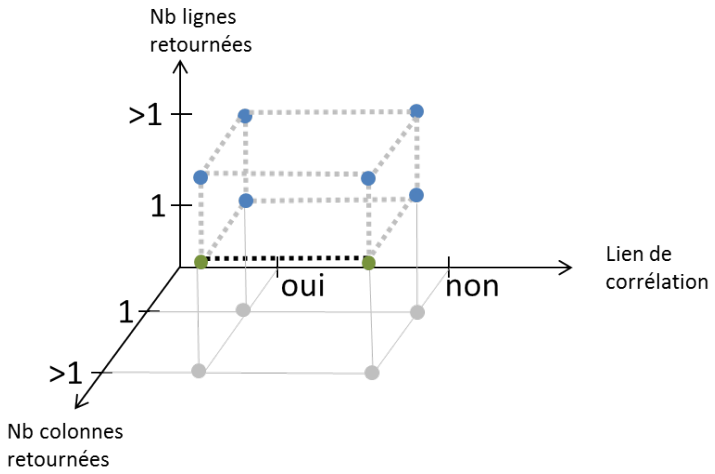
idf	titre	annee	genre	real	budget
1	Edge of tomorrow	2014	SF	7	178
2	Lucy	2014	SF	8	40
3	The Artist	2011	Drame	9	9
4	OSS 117: Le Caire, Nid d'espions	2006	NULL	9	14
5	Bienvenue chez les Ch'tis	2008	Comédie	5	11

Résultat :

prenom	nom
Scarlett	JOHANSSON
Tom	CRUISE
Dany	BOON

Et si la sous-requête retournait deux attributs ?

Les résultat d'une sous-requête qui ne retourne qu'un seul tuple avec plusieurs attributs peut être comparé à un ensemble d'attributs.



Cas où la sous-requête mono-ligne avec colonne double et l'égalité comme comparaison

SELECT *
FROM $nomTable_1$
WHERE $(att_1, att_2) =$ (**SELECT** att_3, att_4
FROM $nomTable_2$) ;

Exemple 2 de sous-requête

Donner le titre des films de même genre et sortie la même année que le film "Lucy".

```
SELECT F.titre
 FROM Film F
 WHERE (F.annee, F.genre) = (SELECT F.annee, F.genre
                             FROM Film F
                             WHERE F.titre = 'Lucy');
```

Résultat :

titre
Edge of tomorrow
Lucy

Remarque: Nous verrons comment faire pour que Lucy n'apparaisse pas dans le résultat en créant une corrélation entre la sous-requête et la requête principale. (*voir les "sous-requêtes corrélées"*).

Et si la sous-requête retournait n attributs ?

La notation se généralise pour n attributs :

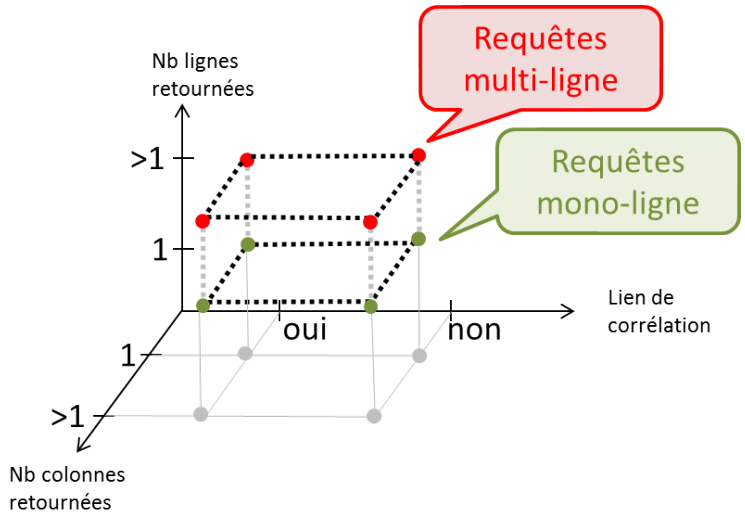
Cas où la sous-requête mono-ligne et multi-colonne avec l'égalité comme comparaison

SELECT *
FROM $nomTable_1$
WHERE
$(att_1, ..., att_n) = (\textbf{SELECT}\ att_1, ..., att_n \textbf{FROM}\ nomTable_2)$;

Il est important que le nombre d'attributs à gauche de l'opérateur soit le même que le nombre d'attributs dans le **SELECT** de la requête imbriquée.

Et si la sous-requête retournait plusieurs tuples ?

Il n'est plus possible de comparer directement un attribut avec le résultat de la sous-requête (qui est un ensemble de valeurs)



Dans ce cas, il est nécessaire d'introduire de nouveaux opérateurs permettant d'introduire la sous-requête dans la requête.
Voyons maintenant les opérateurs **IN** , **ANY** et **ALL** .

attribut **IN** Sous-requête

Pour ajouter une condition d'appartenance d'une valeur d'attribut au résultat d'une requête retournant potentiellement un ensemble de valeurs, il est possible d'utiliser **IN** .

Expression de la contrainte : "mon attribut doit appartenir au résultat de la sous-requête"

SELECT *
FROM $nomTable_1$
WHERE att_1 **IN** (**SELECT** att_3 **FROM** $nomTable_2$) ;

Remarque: Il est important que le nombre d'attributs retournés par la sous-requête corresponde au nombre d'attributs à gauche du **IN** .

attribut **IN** Sous-requête

Car il est également possible d'exprimer la contrainte sur plusieurs attributs

Expression de la contrainte : "mon couple d'attributs doit appartenir au résultat de la sous-requête"

SELECT *
FROM $nomTable_1$
WHERE (att_1, att_2) **IN**
($\mathbf{SELECT}\ att_3, att_4\ \mathbf{FROM}\ nomTable_2$) ;

Exemple 3 de sous-requête

Donner le nom des personnages qui apparaissent dans un film réalisé par 'Michel HAZANAVICIUS'.

```
SELECT JD.nom
FROM JoueDans JD
WHERE JD.idf IN
        (SELECT F.idf
         FROM Film F JOIN Artiste A
         ON F.real = A.ida
         WHERE A.prenom = 'Michel'
         AND A.nom = 'HAZANAVICIUS');
```

Résultat :

nom
George Valentine
Hubert Bonisseur de la Bath
Larmina El Akmar Betouche
Peppy Miller

Question ?

Est-ce que les requêtes ci-dessous retournent forcément le même résultat (*i.e.*, , sont-elles équivalentes quelque soit le jeu de données ?)

```
SELECT *
FROM nomTable1
 WHERE (att1, att2) IN
                (SELECT att3, att4 FROM nomTable2);
```

```
SELECT *
FROM nomTable1
 WHERE att1 IN
                (SELECT att3 FROM nomTable2)
 AND att2 IN
                (SELECT att4 FROM nomTable2);
```

attribut **NOT IN** sous-requête

Il est possible de combiner un **IN** avec un **NOT** pour ajouter une condition de non appartenance d'une valeur d'attribut au résultat d'une sous-requête.

Expression de la contrainte : "mon attribut ne doit pas appartenir au résultat de la sous-requête"

SELECT *
FROM $nomTable_1$
WHERE att_1 **NOT IN** (**SELECT** att_3 **FROM** $nomTable_2$) ;

Exemple 4 de sous-requête

Donner le prénom et le nom des artistes qui n'ont jamais joué dans un film réalisé par 'Doug LIMAN'.

```
SELECT A.nom
FROM Artiste A
WHERE A.ida NOT IN
        (SELECT JD.ida
         FROM JoueDans JD JOIN Film F
         ON JD.idf = F.idf
         JOIN Artiste ON F.real = A.ida
         WHERE A.prenom = 'Doug'
         AND A.nom = 'LIMAN');
```

Résultat :

prenom	nom
Jean	DUJARDIN
Scarlett	JOHANSSON
Bérénice	BEJO
Dany	BOON

Question ?

Que retourne la requête suivante ?

```
SELECT A.nom
FROM Artiste A
WHERE A.idp IN
            (SELECT JD.idp
             FROM JoueDans JD JOIN Film F
             ON JD.idf != F.idf
             JOIN Artiste ON F.real = A.ida
             WHERE A.prenom = 'Doug'
             AND A.nom = 'LIMAN');
```

attribut <comparateur> **ANY** sous-requête

Pour ajouter une condition permettant de vérifier qu'une comparaison est vraie au moins un fois entre un attribut et l'ensemble des valeurs retournées par une sous-requête, il est possible d'utiliser $\square$ **ANY** , avec , $\square \in \{=, !=, <, =<, =>, >\}$.

Expression de la contrainte : "La valeur de l'attribut att_1 est inférieurs strictement à au moins une des valeurs de att_3 retournées par la sous-requête"

SELECT *
FROM $nomTable_1$
WHERE att_1 $<$ **ANY** (**SELECT** att_3 **FROM** $nomTable_2$) ;

Remarque: Il est important que le types des attributs (ici att_1 et att_3) soient les mêmes pour pouvoir appliquer la comparaison.

Exemple 5 de sous-requête

Donner le titre des films dont le budget est supérieur strictement à au moins un des films réalisé par 'Michel HAZANAVICIUS'.

```
SELECT Fr.titre
FROM Film Fr
WHERE Fr.budget > ANY
            (SELECT F.budget
             FROM Film F JOIN Artiste A
             ON F.real = A.ida
             WHERE A.prenom = 'Michel'
             AND A.nom = 'HAZANAVICIUS');
```

Résultat :

titre
Edge of tomorrow
Lucy
OSS 117 : Le Caire, Nid d'espions
Bienvenue chez les Ch'tis

Remarque: Nous aurons besoin de modifier la requête pour ne pas avoir OSS 117 dans le résultats (*voir les "sous-requêtes corrélées"*).

attribut <comparateur> **ALL** sous-requête

Pour ajouter une condition permettant de vérifier qu'une comparaison toujours est vraie entre un attribut et l'ensemble des valeurs retournées par une sous-requête, il est possible d'utiliser $\square$ **ALL** , avec , $\square \in \{=, !=, <, =<, =>, >\}$.

Expression de la contrainte : "La valeur de l'attribut att_1 est inférieurs strictement à toutes les valeurs de att_3 retournées par la sous-requête"

SELECT *
FROM $nomTable_1$
WHERE att_1 $<$ **ALL** (**SELECT** att_3 **FROM** $nomTable_2$) ;

Remarque: Il est important que le types des attributs (ici att_1 et att_3) soient les mêmes pour pouvoir appliquer la comparaison.

Exemple 6 de sous-requête

Donner le titre des films dont le budget est supérieur strictement au budget de n'importe quel film réalisé par 'Michel HAZANAVICIUS'.

```
SELECT Fr.titre
FROM Film Fr
WHERE Fr.budget > ALL
        (SELECT F.budget
         FROM Film F JOIN Artiste A
         ON F.real = A.ida
         WHERE A.prenom = 'Michel'
         AND A.nom = 'HAZANAVICIUS');
```

Résultat :

titre
Edge of tomorrow
Lucy

Question ?

Que retourne la requête suivante ?

```
SELECT Fr.titre
FROM Film Fr
WHERE Fr.budget <= ALL (SELECT F.budget
                        FROM Film F );
```

Retour sur l'exemple 2 de sous-requête

Donner le titre des films de même genre et sortie la même année que le film "Lucy".

```
SELECT F.titre
 FROM Film F
 WHERE (F.annee, F.genre) = (SELECT F.annee, F.genre
                             FROM Film F
                             WHERE F.titre = 'Lucy');
```

Résultat :

titre
Edge of tomorrow
Lucy

Retour sur l'exemple 2 de sous-requête

Pour résoudre le problème, il serait nécessaire de préciser dans la requête principale que les titres de film retournés ne doivent pas inclure 'Lucy'.
Une façon de faire serait d'ajouter une condition dans le **WHERE** de la requête principale.

Donner le titre des films de même genre et sortie la même année que le film "Lucy".

```
SELECT F.titre
 FROM Film F
 WHERE F.titre != 'Lucy'
 AND (F.annee, F.genre) = (SELECT F.annee, F.genre
                           FROM Film F
                           WHERE F.titre = 'Lucy');
```

Résultat :

titre
Edge of tomorrow

Retour sur l'exemple 2 de sous-requête

Une autre façon de faire, serait de transformer la sous-requête en sous-requête corrélée. L'instance de Film dans la requête principale peut être utilisée dans la sous-requête.

Donner le titre des films de même genre et sortie la même année que le film "Lucy".

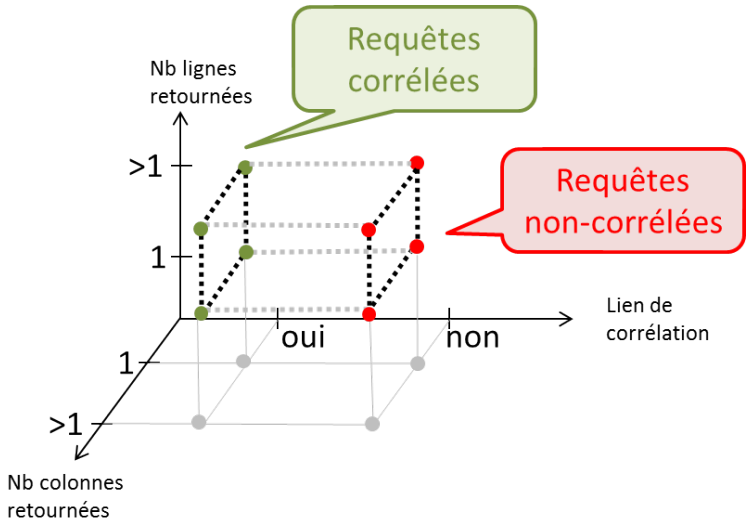
```
SELECT F.titre
 FROM Film F
 WHERE (F.annee, F.genre) = (SELECT Fl.annee, Fl.genre
                             FROM Film Fl
                             WHERE Fl.titre = 'Lucy'
                             AND Fl.idf != F.idf) ;
```

Résultat :

titre
Edge of tomorrow

Les sous-requêtes corrélées

Une sous-requête qui contient une référence (via un alias) à une instance de la requête principale est nommée : **Sous-requête corrélée**.



Expression d'une requête corrélée

SELECT *
FROM $nomTable_1$ T1
WHERE att_1 **IN** (**SELECT** att_3
FROM $nomTable_2$ $T2$
WHERE $T2.att_4 = T1.att_2$);

Les sous-requêtes corrélées

Expression d'une requête corrélée

SELECT *
FROM $nomTable_1$ T1
WHERE att_1 **IN** (**SELECT** att_3
FROM $nomTable_2\ T2$
WHERE $T2.att_4 = T1.att_2$);

Remarque: Il est important de comprendre que contrairement à T1 qui apparaît dans la sous-requête, T2 ne pourrait pas apparaître hors de la sous-requête.

EXISTS (sous-requête corrélée)

Pour ajouter une condition portant non pas que les valeurs retournées par la sous-requête mais sur l'existence ou non de tuples vérifiant la sous-requête, il est possible d'utiliser **EXISTS** et le plus souvent la combinaison **NOT EXISTS** .

Expression de la contrainte : "La valeur de l'attribut att_1 est inférieurs strictement à au moins une des valeurs de att_3 retournées par la sous-requête"

```
SELECT *
FROM nomTable1 T1
 WHERE EXISTS (SELECT *
                    FROM nomTable2 T2)
                    WHERE T1.att4 = T2.att2);
```

Remarque: Il est important de noter qu'il n'y a pas d'attribut à la gauche du mot **EXISTS** .

Exemple 7 de sous-requête

Donner le nom des réalisateurs qui n'ont jamais fait jouer 'Jean DUJARDIN' dans un de leur film.

```
SELECT Ar.nom
FROM Artiste Ar
WHERE NOT EXISTS (SELECT F.*
                  FROM Film F JOIN JoueDans JD
                  ON F.idf = JD.idf
                  JOIN Artiste A ON A.idp = JD.idp
                  WHERE Ar.idp = F.real
                  AND A.prenom = 'Jean'
                  AND A.nom = 'DUJARDIN');
```

Résultat :

nom
BESSON
LIMAN

Question ?

Que retourne la requête suivante ?

```
SELECT A.prenom, A.nom
FROM Artiste A
WHERE NOT EXISTS (SELECT F.*
                  FROM Film F JOIN Artiste Ar
                  ON F.real = Ar.idp
                  WHERE Ar.prenom = 'Michel'
                  AND Ar.nom = 'HAZANAVICIUS'
                  AND NOT EXISTS (SELECT JD.*
                                  FROM JoueDans JD
                                  WHERE JD.idf = F.idf
                                  AND JD.idp = A.idp ));
```

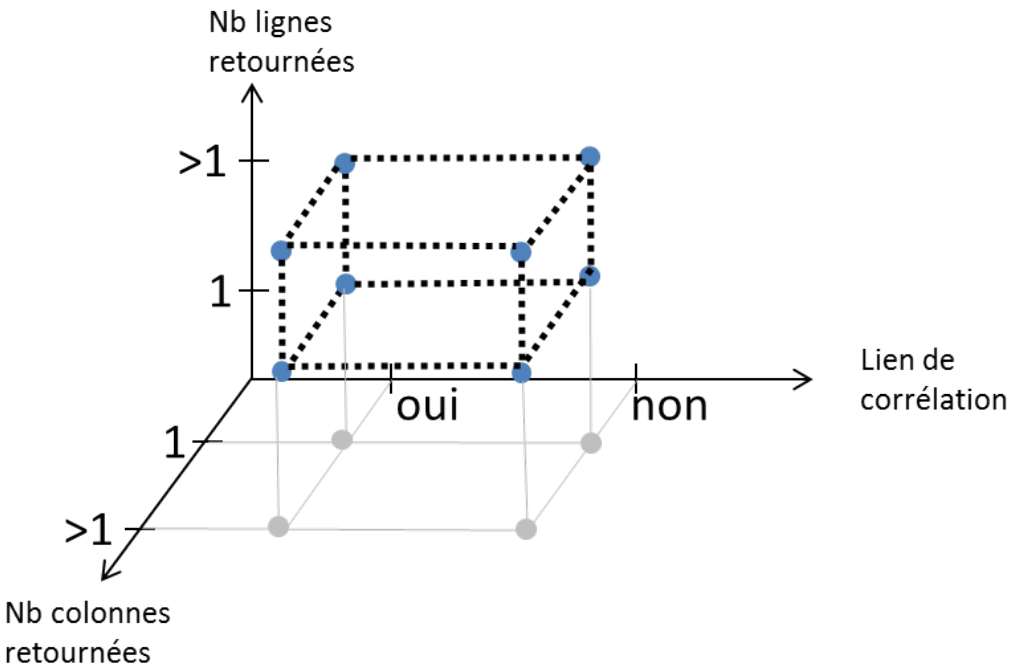
Sous-requête imbriquée

Est-ce qu'une sous-requête peut elle-même une sous-requête ?
Oui ! nous parlons alors de **sous-requêtes imbriquées**

Expression de sous-requêtes imbriquées

SELECT *
FROM $nomTable_1$ T1
WHERE $T1.att_1$ **IN**
(**SELECT** att_2
FROM $nomTable_2$ $T2$
WHERE $T2.att_3$ **IN**
(**SELECT** att_4
FROM $nomTable_3$ $T3$
WHERE ...)));

Bilan sur les sous-requêtes



- Bien réfléchir au résultat retourné par une sous-requête (nombre de lignes, de colonnes)
- Utiliser un comparateur ou un opérateur approprié selon ce résultat

Plan

Remarque

Certains besoins d'utilisateurs peuvent s'exprimer de la façon suivante :

Exemples

- Donner les noms de tous les acteurs et les noms des rôles de films de genre 'SF'.
- Donner le nom de tous les acteurs/actrices.
- Donner le nom des acteurs-réalisateurs.

Opérateur d'union de requêtes : UNION

Syntaxe d'une union de requêtes est :

(**SELECT** att_1 , att_2 ...
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
...)
UNION
(**SELECT** att_3 , att_4 ...
FROM $nomTable_2$
WHERE $< condition\_de\_selection >$
...) ;

- Le **UNION** permet d'unir le résultat de deux requêtes SQL
- Les clauses **SELECT** des deux requêtes doivent retourner le même nombre d'attributs et des attributs de même types

Exemple d'**UNION**

Donner les noms de tous les acteurs/actrices et les noms des rôles de films de genre 'SF'.

```
(SELECT A.nom
FROM Artiste A
WHERE A.ida NOT IN
        (SELECT real FROM Film)
) UNION
(SELECT JD.nom
FROM JoueDans JD JOIN Film F
ON JD.idf = F.idf
WHERE F.genre = 'SF');
```

Opérateur d'intersection de requêtes : INTERSECT

Syntaxe d'une intersection de requêtes est :

(SELECT att_1 , att_2 ...
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
...)
INTERSECT
(SELECT att_3 , att_4 ...
FROM $nomTable_2$
WHERE $< condition\_de\_selection >$
...);

- Le **INTERSECT** permet de retourner les tuples communs à deux requêtes SQL.
- Les clauses **SELECT** des deux requêtes doivent retourner le même nombre d'attributs et des attributs de même types.

Opérateur d'intersection de requêtes : INTERSECT

Remarque:
L'opérateur **INTERSECT** n'est pas implémenté dans MySQL ! **Il est possible de le remplacer par une jointure.**

Exemple d'INTERSECT

Donner le nom des acteurs/actrices qui sont aussi réalisateurs/trices .

```
(SELECT A.nom
FROM Artiste A
WHERE A.ida NOT IN
      (SELECT real FROM Film)
) INTERSECT
(SELECT JD.nom
FROM JoueDans JD JOIN Film F
ON JD.idf = F.idf
WHERE F.genre = 'SF');
```

Opérateur de différence entre des requêtes : MINUS

Syntaxe d'une intersection de requêtes est :

```
(SELECT att1 , att2 ...
FROM nomTable1
WHERE < condition_de_selection >
...)
MINUS
(SELECT att3 , att4 ...
FROM nomTable2
WHERE < condition_de_selection >
...);
```

- Le **MINUS** permet de retirer du résultat de la première requête les tuples communs avec la seconde requête.
- Les clauses **SELECT** des deux requêtes doivent retourner le même nombre d'attributs et des attributs de même types.

Opérateur de différence entre des requêtes : **MINUS**

Remarque:
L'opérateur **MINUS** n'est pas implémenté dans MySQL ! **Il est possible de le remplacer par une sous-requête avec une NOT IN .**

Exemple de **MINUS**

Donner le nom de tous les acteurs/actrices.

```
(SELECT A.nom
FROM Artiste A
WHERE A.ida NOT IN
      (SELECT real FROM Film)
) MINUS
(SELECT JD.nom
FROM JoueDans JD JOIN Film F
ON JD.idf = F.idf
WHERE F.genre = 'SF');
```

Plan

Remarque

Certains besoins d'utilisateurs peuvent s'exprimer de la façon suivante :

Exemples

- Pour chaque artiste, donner le nombre de films dans lequel il/elle a joué depuis 2010.
- Pour chaque film, donner le salaire moyen des artistes associés au film.
- Pour chaque film, donner le montant du budget consacré au salaire des acteurs/actrices.

Ce que nous avons appris de la syntaxe SQL jusqu'à présent, ne nous permet pas de répondre à ces besoins.

Intuition

Pour chaque artiste, donner le nombre de films dans lequel il/elle a joué depuis 2010.

Il faudrait pouvoir regrouper les tuples en fonction de l'identifiant des artistes et compter le nombre d'identifiants de film qui apparaissent et ce, pour chaque groupe de tuples.

Pour chaque film, donner le salaire moyen des artistes associés au film.

Il faudrait pouvoir regrouper les tuples en fonction de l'identifiant des films, lister les salaires associés à chaque film et calculer la moyenne des salaires présents dans chaque groupe de tuples.

$\Rightarrow$ Apprenons a construire des groupes de tuples !

Clause de regroupement : GROUP BY

Syntaxe d'une requête SQL avec regroupement :

SELECT att_1 , att_2 ...
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
GROUP BY att_i, att_j ;

Le **GROUP BY** , exécuté après le **WHERE** , indique de procéder à une répartition du résultat en groupes de tuples :

- Deux tuples sont dans un groupe s'il ont mêmes valeurs sur les attributs spécifiés dans le **GROUP BY** (*i.e.*, att_i, att_j)
- Si deux tuplets sont dans deux groupes différents, alors il y a au moins un attribut parmi ceux spécifiés dans le **GROUP BY** pour lequel ils ont une valeur différente.

Questions !

- Combien de tuples sont retournés par groupe dans le résultat ?

  La requête ne renvoie **qu'un seul tuple par groupe.**

- Les attributs présents dans le **SELECT** et le **ORDER BY** ont-il un lien avec les attributs présents dans le **GROUP BY** ?

  OUI ! Le **SELECT** et le **ORDER BY** ne peuvent utiliser que des attributs présents dans le **GROUP BY** .
  - Dans un groupe, la valeur pour les attributs du **GROUP BY** est fixe,
  - Mais la valeur pour les autres attributs peut varier, d'où leur utilisation directe impossible (quelle valeur utiliser ?)*

*En fait, pour pouvoir les utiliser, il sera nécessaire d'utiliser des fonctions de regroupements

Questions !

- Combien de tuples sont retournés par groupe dans le résultat ?

  La requête ne renvoie **qu'un seul tuple par groupe.**

- Les attributs présents dans le **SELECT** et le **ORDER BY** ont-il un lien avec les attributs présents dans le **GROUP BY** ?

  OUI ! Le **SELECT** et le **ORDER BY** ne peuvent utiliser que des attributs présents dans le **GROUP BY** .
  - Dans un groupe, la valeur pour les attributs du **GROUP BY** est fixe,
  - Mais la valeur pour les autres attributs peut varier, d'où leur utilisation directe impossible (quelle valeur utiliser ?)*

*En fait, pour pouvoir les utiliser, il sera nécessaire d'utiliser des fonctions de regroupements

Exemple 1 de **GROUP BY**

Donner les nationalités, autre qu'américaines, représentées parmi les artistes.

```
SELECT A. nation
FROM Artiste A
WHERE A. nation != 'USA'
GROUP BY A. nation ;
```

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Résultat :

nation
FR
FR-AR
UK

Comment s'est exécutée la requête ?

```
SELECT A. nation
FROM Artiste A
WHERE A. nation != 'USA'
GROUP BY A. nation ;
```

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Étape 1 - Application des conditions de sélection :
"...**WHERE** *A*. nation != 'USA'".

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Comment s'est exécutée la requête ?

```
SELECT A. nation
FROM Artiste A
WHERE A. nation != 'USA'
GROUP BY A. nation ;
```

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Étape 2 - Application du regroupement :
"... **GROUP BY** *A*. nation"

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
5	Dany	BOON	FR
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR
4	Bérénice	BEJO	FR-AR
6	Emily	BLUNT	UK

Comment s'est exécutée la requête ?

```
SELECT A. nation
FROM Artiste A
WHERE A. nation != 'USA'
GROUP BY A. nation ;
```

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Étape 3 - Application de la projection :
"**SELECT** *A*. nation"

nation
FR
FR
FR
FR
FR-AR
UK

Comment s'est exécutée la requête ?

```
SELECT A. nation
FROM Artiste A
WHERE A. nation != 'USA'
GROUP BY A. nation ;
```

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Étape 4 - Fusion des résultats pour ne retourner qu'un tuple par groupe

nation
FR
FR-AR
UK

Fonction d'agrégation

Fonctions d'agrégation (ou opérateurs de regroupements) :

- Fonctions permettant :
  - de dénombrer des tuples (*count*)
  - de calculer une moyenne (*avg*), une somme cumulée (*sum*), un minimum (*min*), un maximum (*max*), ...
- Ces fonctions d'agrégation :
  - sont utilisables en conjonction avec un **GROUP BY** pour s'appliquer sur les attributs qui ne font pas partie du **GROUP BY**
  - apparaissent pour l'instant uniquement dans le **SELECT** et dans le **ORDER BY**

Remarque: On ne peut pas les utiliser dans le **WHERE** , car comme nous l'avons vu dans le déroulement de l'exécution de la requête le **WHERE** a lieu avant regroupement

Remarque sur les fonctions d'agrégations

A propos des fonctions d'agrégation :

- Il est possible d'utiliser des opérateurs d'agrégation sans **GROUP BY** . Si aucun **GROUP BY** n'est spécifié dans la requête, l'opérateur s'applique à toutes les lignes renvoyées par la requête.

- Il est possible de placer le mot clé **DISTINCT** dans la fonction de regroupement (avant l'attribut) pour appliquer la fonction d'agrégation en considérant ou non les doublons

Opérateur COUNT

count(att_i) : le nombre d'occurrences de att_i dans le groupe

SELECT att_1 , COUNT(att_2) **AS** NB
FROM $nomTable_1$
WHERE $<$ *condition_de_selection* $>$
GROUP BY att_1 ;

Remarque:

- Les n-uplets pour lesquels e vaut NULL ne sont pas comptés
- Quand il ne s'agit pas de dénombrer des valeurs mais uniquement les tuples composant chaque groupe, il est possible d'écrire COUNT($*$).

Exemple 1 d'opérateur COUNT

Pour chaque nationalité, donner le nombre d'artistes ayant cette nationalité.

```
SELECT A. nation, COUNT(*) AS NbArtistes
FROM Artiste A
GROUP BY A. nation ;
```

Artiste(ida, nom, prenom, nation)

ida	prenom	nom	nation
1	Jean	DUJARDIN	FR
2	Scarlett	JOHANSSON	USA
3	Tom	CRUISE	USA
4	Bérénice	BEJO	FR-AR
5	Dany	BOON	FR
6	Emily	BLUNT	UK
7	Doug	LIMAN	USA
8	Luc	BESSON	FR
9	Michel	HAZANAVICIUS	FR

Résultat :

nation	NbArtistes
FR	4
USA	3
FR-AR	1
UK	1

Exemple 2 d'opérateur COUNT

Donner le nombre d'années de sortie de film représentées dans la base.

```
SELECT COUNT(DISTINCT annee) AS NbAnnees
FROM Film;
```

Film(idf, titre, annee, genre, real, budget)

idf	titre	annee	genre	real	budget
1	Edge of tomorrow	2014	SF	7	178
2	Lucy	2014	SF	8	40
3	The Artist	2011	Drame	9	9
4	OSS 117: Le Caire, Nid d'espions	2006	NULL	9	14
5	Bienvenue chez les Ch'tis	2008	Comédie	5	11

Résultat :

NbAnnees
4

Exemple 3 d'opérateur COUNT

Pour chaque acteur/actrice français(e) dont on précisera le nom et le prénom, donner le nombre de films dans lequel il/elle a joué. Le résultat du calcul sera nommé "NbFilms" et le résultat retourné sera trié par ordre lexicographique sur le nom et le prénom des artistes

```
SELECT A.prenom, A.nom, COUNT(JD.idf) AS NbFilms
FROM Artiste A JOIN JoueDans JD ON A.idp = JD.idp
WHERE A.nation LIKE '%FR%'
GROUP BY A.prenom, A.nom, A.idp
ORDER BY A.nom, A.prenom ;
```

Résultat :

prenom	nom	NbFilms
Bérénice	BEJO	2
Dany	BOON	1
Jean	DUJARDIN	2

Comment s'est exécutée la requête ?

```
SELECT A.prenom, A.nom, COUNT(JD.idf) AS NbFilms
FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
WHERE A.nation LIKE '%FR%'
GROUP BY A.prenom, A.nom, A.ida
ORDER BY A.nom, A.prenom;
```

Étape 1 - Exécution de la jointure

ida	prenom	Artiste.nom	nation	idf	JoueDans.nom	sal
1	Jean	DUJARDIN	FR	3	George Valentin	0,3
1	Jean	DUJARDIN	FR	4	Hubert Bonisseur de La Bath	0,2
2	Scarlett	JOHANSSON	USA	2	Lucy Miller	14
3	Tom	CRUISE	USA	1	Bill Cage	20
4	Bérénice	BEJO	FR-AR	4	Larmina El Akmar Betouche	NULL
4	Bérénice	BEJO	FR-AR	3	Peppy Miller	NULL
5	Dany	BOON	FR	5	Antoine Bailleul	1,3
6	Emily	BLUNT	UK	1	Rita Vrataski	NULL

Comment s'est exécutée la requête ?

```
SELECT A.prenom, A.nom, COUNT(JD.idf) AS NbFilms
FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
WHERE A.nation LIKE '%FR%'
GROUP BY A.prenom, A.nom, A.ida
ORDER BY A.nom, A.prenom;
```

Étape 2 - Exécution du **WHERE**

ida	prenom	Artiste.nom	nation	idf	JoueDans.nom	sal
1	Jean	DUJARDIN	FR	3	George Valentin	0,3
1	Jean	DUJARDIN	FR	4	Hubert Bonisseur de La Bath	0,2
4	Bérénice	BEJO	FR-AR	4	Larmina El Akmar Betouche	NULL
4	Bérénice	BEJO	FR-AR	3	Peppy Miller	NULL
5	Dany	BOON	FR	5	Antoine Bailleul	1,3

Comment s'est exécutée la requête ?

```
SELECT A.prenom, A.nom, COUNT(JD.idf) AS NbFilms
FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
WHERE A.nation LIKE '%FR%'
GROUP BY A.prenom, A.nom, A.ida
ORDER BY A.nom, A.prenom;
```

Étape 3 - Exécution du **GROUP BY**

ida	prenom	Artiste.nom	nation	idf	JoueDans.nom	sal
1	Jean	DUJARDIN	FR	3	George Valentin	0,3
1	Jean	DUJARDIN	FR	4	Hubert Bonisseur de La Bath	0,2
4	Bérénice	BEJO	FR-AR	4	Larmina El Akmar Betouche	NULL
4	Bérénice	BEJO	FR-AR	3	Peppy Miller	NULL
5	Dany	BOON	FR	5	Antoine Bailleul	1,3

Comment s'est exécutée la requête ?

```
SELECT A.prenom, A.nom, COUNT(JD.idf) AS NbFilms
FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
WHERE A.nation LIKE '%FR%'
GROUP BY A.prenom, A.nom, A.ida
ORDER BY A.nom, A.prenom;
```

Étape 4 - Exécution du **ORDER BY**

ida	prenom	Artiste.nom	nation	idf	JoueDans.nom	sal
4	Bérénice	BEJO	FR-AR	4	Larmina El Akmar Betouche	NULL
4	Bérénice	BEJO	FR-AR	3	Peppy Miller	NULL
5	Dany	BOON	FR	5	Antoine Bailleul	1,3
1	Jean	DUJARDIN	FR	3	George Valentin	0,3
1	Jean	DUJARDIN	FR	4	Hubert Bonisseur de La Bath	0,2

Comment s'est exécutée la requête ?

```
SELECT A.prenom, A.nom, COUNT(JD.idf) AS NbFilms
FROM Artiste A JOIN JoueDans JD ON A.ida = JD.ida
WHERE A.nation LIKE '%FR%'
GROUP BY A.prenom, A.nom, A.ida
ORDER BY A.nom, A.prenom;
```

Étape 5 - Exécution du **SELECT**

prenom	nom	NbFilms
Bérénice	BEJO	2
Dany	BOON	1
Jean	DUJARDIN	2

Opérateurs *min* et *max*

- $min(att_i)$: la valeur minimale de l'attribut att_i pour le groupe
- $max(att_i)$: la valeur maximale de l'attribut att_i pour le groupe

SELECT att_1 , $min(att_2)$ **AS** MinVal
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
GROUP BY att_1 ;

SELECT att_1 , $max(att_2)$ **AS** MaxVal
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
GROUP BY att_1 ;

Exemple pour l'opérateur *max*

Pour chaque année, donner le budget du film ayant eu le plus gros budget cette année-là. La valeur s'affichera sous le nom 'LePlusCher' et le résultat sera trié par ordre décroissante des années.

```
SELECT F.annee, MAX(f.budget) AS LePlusCher
FROM Film F
GROUP BY F.annee
ORDER BY F.annee DESC ;
```

Résultat :

annee	LePlusCher
2014	178
2011	9
2008	11
2006	14

Exemple pour l'opérateur *min*

Pour chaque film dont on précisera le titre, donner le nom et le prénom des artistes qui ont reçu le plus petit salaire pour le film.

```
SELECT F.titre, A.prenom, A.nom
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
JOIN Artiste F ON A.ida = JD.ida
WHERE JD.sal =
                    (SELECT min(JD2.sal)
                    FROM JoueDans JD2
                    WHERE JD2.idf = F.idf)
```

Résultat :

titre	prenom	nom
Edge of tomorrow	Tom	CRUISE
Lucy	Scarlett	JOHANSSON
The Artist	Jean	DUJARDIN
OSS 117 : Le Caire, Nid d'espions	Jean	DUJARDIN
Bienvenue chez les Ch'tis	Dany	BOON

Opérateurs *avg* et *sum*

- $avg(att_i)$: la moyenne de l'attribut att_i pour le groupe
- $sum(att_i)$: la somme cumulée de l'attribut att_i pour le groupe

SELECT att_1 , $avg(att_2)$ **AS** MinVal
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
GROUP BY att_1 ;

SELECT att_1 , $sum(att_2)$ **AS** MaxVal
FROM $nomTable_1$
WHERE $< condition\_de\_selection >$
GROUP BY att_1 ;

Exemple pour l'opérateur *avg*

Pour chaque film dont on précisera le titre, donner salaire moyen des artistes associés au film. La valeur calculée s'affichera sous le nom 'MoySal'

SELECT *F*.titre, *avg*(*JD*.sal) **AS** MoySal
FROM Film *F* **JOIN** JoueDans *JD* **ON** *F*.idf = *JD*.idf
GROUP BY *F*.titre, *F*.idf

Résultat :

titre	MoySal
Edge of tomorrow	20
Lucy	14
The Artist	0.3
OSS 117 : Le Caire, Nid d'espions	0.2
Bienvenue chez les Ch'tis	1.3

Exemple pour l'opérateur *sum*

Pour chaque film dont on précisera le titre, donner le montant du budget non consacré au salaire des acteurs/actrices. La valeur calculée s'affichera sous le nom 'NonSal'

SELECT *F*.titre, *F*.budget -*sum*(*JD*.sal) **AS** NonSal
FROM Film *F* **JOIN** JoueDans *JD* **ON** *F*.idf = *JD*.idf
GROUP BY *F*.titre, *F*.budget, *F*.idf

Résultat :

titre	NonSal
Edge of tomorrow	20
Lucy	14
The Artist	0.3
OSS 117 : Le Caire, Nid d'espions	0.2
Bienvenue chez les Ch'tis	1.3

Plan du cours

Remarque

Certains besoins d'utilisateurs peuvent également s'exprimer de la façon suivante :

Exemples

- Donner les années pour lesquelles au moins deux films sont sortis.
- Donner le titre des films dont le salaire moyen des acteurs est supérieur à 1.
- Donner le titre des films dont au moins deux acteurs ont un salaire supérieur à 0.5.

Ce dont nous avons besoin, c'est de pouvoir exprimer des conditions de sélection au niveau des groupes.

Sélection sur les groupes

Syntaxe d'une requête SQL avec sélection sur les groupes :

SELECT att_i , att_j , ...
FROM $nomTable_1$
[**WHERE** $<$ *condition_de_selection_de_tuples* $>$]
GROUP BY att_i , att_j , ...
HAVING $<$ *condition_de_selection_de_groupe* $>$;

- Clause **HAVING** permet de sélectionner les groupes qui satisfont la condition
  $<$ *condition_de_selection_de_groupe* $>$

Remarque

- Il est important de distinguer les clauses **WHERE** et **HAVING**
  - La clause **WHERE** s'applique sur les tuples de manière individuelle,
    et ce avant le regroupement fait par le **GROUP BY** .
  - La clause **HAVING** s'applique sur les groupes et non pas sur les tuples individuels.

  $\Rightarrow$ Il n'y aura **JAMAIS** d'opérateurs d'agrégation directement dans un **WHERE** .
- La clause **HAVING** est exécutée entre le **GROUP BY** et le **ORDER BY** .
- Le **HAVING** peut porter sur :
  - les attributs spécifiés dans le **GROUP BY**
  - le résultat d'opérateurs d'agrégation (appliqués sur des attributs qui ne sont pas spécifié dans le **GROUP BY**)

Exemple 1 pour le **HAVING**

Donner les années pour lesquelles au moins deux films sont sortis

```
SELECT F.annee
FROM Film F
GROUP BY F.annee
HAVING count(*) >= 2;
```

Film(idf, titre, annee, genre, real, budget)

idf	titre	annee	genre	real	budget
1	Edge of tomorrow	2014	SF	7	178
2	Lucy	2014	SF	8	40
3	The Artist	2011	Drame	9	9
4	OSS 117: Le Caire, Nid d'espions	2006	NULL	9	14
5	Bienvenue chez les Ch'tis	2008	Comédie	5	11

Résultat :

annee
2014

Exemple 2 pour le **HAVING**

Donner le titre des films sortis depuis 2010 dont le salaire moyen des acteurs est supérieur strictement à 1 et préciser cette valeur de salaire moyen en l'affichant sous le nom 'MoySalSup1'. Le résultat sera trié sur le titre selon l'ordre lexicographique inverse.

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Résultat :

titre	MoySalSup1
Lucy	14
Edge of tomorrow	20

Comment s'est exécutée la requête ?

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Étape 1 - Exécution de la jointure

idf	titre	annee	genre	real	budget	ida	nom	sal
1	Edge of tomorrow	2014	SF	7	178	3	Bill Cage	20
1	Edge of tomorrow	2014	SF	7	178	6	Rita Vrataski	NULL
2	Lucy	2014	SF	8	40	2	Lucy Miller	14
3	The Artist	2011	Drame	9	9	1	George Valentin	0.3
3	The Artist	2011	Drame	9	9	4	Peppy Miller	NULL
4	OSS 117 : Le Caire ...	2006	NULL	9	14	1	Hubert Bonisseur ...	0.2
4	OSS 117 : Le Caire ...	2006	NULL	9	14	4	Larmina El Akmar ...	NULL
5	Bienvenue chez les Ch'tis	2008	Comédie	5	11	5	Antoine Bailleul	1.3

Comment s'est exécutée la requête ?

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Étape 2 - Exécution du **WHERE**

idf	titre	annee	genre	real	budget	ida	nom	sal
1	Edge of tomorrow	2014	SF	7	178	3	Bill Cage	20
1	Edge of tomorrow	2014	SF	7	178	6	Rita Vrataski	NULL
2	Lucy	2014	SF	8	40	2	Lucy Miller	14
3	The Artist	2011	Drame	9	9	1	George Valentin	0.3
3	The Artist	2011	Drame	9	9	4	Peppy Miller	NULL

Comment s'est exécutée la requête ?

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Étape 3 - Exécution du **GROUP BY**

idf	titre	annee	genre	real	budget	ida	nom	sal
1	Edge of tomorrow	2014	SF	7	178	3	Bill Cage	20
1	Edge of tomorrow	2014	SF	7	178	6	Rita Vrataski	NULL
2	Lucy	2014	SF	8	40	2	Lucy Miller	14
3	The Artist	2011	Drame	9	9	1	George Valentin	0.3
3	The Artist	2011	Drame	9	9	4	Peppy Miller	NULL

Comment s'est exécutée la requête ?

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Étape 4 - Exécution du **HAVING**

idf	titre	annee	genre	real	budget	ida	nom	sal
1	Edge of tomorrow	2014	SF	7	178	3	Bill Cage	20
1	Edge of tomorrow	2014	SF	7	178	6	Rita Vrataski	NULL
2	Lucy	2014	SF	8	40	2	Lucy Miller	14

Comment s'est exécutée la requête ?

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Étape 4 - Exécution du **ORDER BY**

idf	titre	annee	genre	real	budget	ida	nom	sal
2	Lucy	2014	SF	8	40	2	Lucy Miller	14
1	Edge of tomorrow	2014	SF	7	178	3	Bill Cage	20
1	Edge of tomorrow	2014	SF	7	178	6	Rita Vrataski	NULL

Comment s'est exécutée la requête ?

```
SELECT F.titre, avg(JD.sal) AS MoySalSup1
FROM Film F JOIN JoueDans JD ON F.idf = JD.idf
WHERE F.annee >= 2010
GROUP BY F.titre, F.idf
HAVING MoySalSup1 > 1
ORDER BY F.titre DESC;
```

Étape 5 - Exécution du **SELECT**

titre	MoySalSup1
Lucy	14
Edge of tomorrow	20