Bases de données : Langage SQL (SELECT)

1. Introduction

L'interrogation d'une base de données se fait à l'aide de la commande SQL **SELECT**. Cette commande, appelée également requête de sélection, est fondamentale car elle permet le filtrage et l'analyse des données, et par conséquence d'en extraire des informations utiles.

Les types de requêtes de sélection **SELECT** les plus courantes incluent et combinent :

- **La projection** : la recherche concerne quelques colonnes d'une table.
- **La sélection** : la recherche se rapporte aux lignes d'une table qui vérifient certains critères.
- **La jointure** : le recherche s'adresse aux données de plusieurs tables en relation entre elles.
- **L'agrégation** : les données sont regroupées en fonction des valeurs d'un ou de plusieurs champs dans le but d'effectuer des calculs sur ces groupes.

Les exemples de ce cours utilisent la base de données "**films**" dont la représentation graphique est donnée en figure 1. Les données de ces tables sont données en annexe.

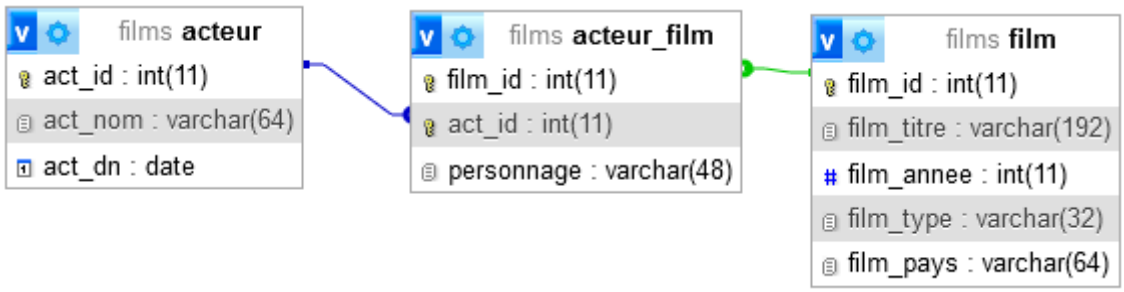


*Figure 1: Représentation graphique de la base de données **films***

2. Projection

Une projection est un type de sélection où seulement une partie (ou la totalité) des attributs d'une table choisie est retenue pour le résultat. La syntaxe générale de la commande SQL est la suivante :

```
SELECT [DISTINCT] * | colonne₁, colonne₂, …
FROM nom_table;
```

Pour afficher, par exemple, les types des films on peut écrire la requête suivante :

```
SELECT film_type FROM film;
```

Uniquement la colonne indiquée (**film_type**) sera affichée.

film_id	film_titre	film_annee	film_type	film_pays
1	Matrix	1999	**Science-fiction**	USA
2	Fast and Furious	2001	**Aventure**	USA
3	John Wick	2014	**Action**	USA
4	Matrix Reloaded	2003	**Science-fiction**	USA

⇒

film_type
Science-fiction
Aventure
Action
Science-fiction

On remarque que le type "**Science-fiction**" s'affiche plusieurs fois.

Pour supprimer les redondances de données on peut écrire :

genre
Science-fiction
Aventure
Action

```
SELECT DISTINCT film_type AS genre
FROM film;
```

Le mot clé **AS** est utilisé pour donner un alias **genre** à la colonne **film_type**. Cette pratique est appréciée pour augmenter la lisibilité des entêtes des requêtes et des colonnes dans le jeu de résultats.

Autres Exemples :

Exemple 1 : Afficher la liste de tous les acteurs de la table **acteur**.

```
SELECT *
FROM acteur;
```

Exemple 2 : Afficher les titres et l'année de lancement de tous les films.

```
SELECT film_titre, film_annee
FROM film;
```

Exemple 3 : Afficher les titres et l'année de lancement des films (2e méthode).

```
SELECT film_titre as titre,
       film_annee as année
FROM film;
```

Exemple 4 : Afficher les pays qui ont réalisé des films sans redondances (sans répétition).

```
SELECT DISTINCT film_pays pays
FROM film;
```

3. Sélection

L'opération de **sélection** est utilisée pour filtrer les données d'une table pour n'afficher que les lignes qui répondent à certains critères. Le filtrage des données est possible en langage SQL grâce à la clause **WHERE**. La syntaxe de la commande SQL devient :

```
SELECT [DISTINCT] * | colonne1, colonne2, …
FROM nom_table
WHERE condition;
```

La **condition** de la clause **WHERE condition**, dans une requête SQL, est une **expression booléenne**. Si le résultat de l'évaluation de cette expression est vrai, pour une ligne, celle-ci fera partie du résultat.

Les opérateurs usuels utilisables dans une expression booléenne sont présentés en annexe 2.

Cette syntaxe combine l'opération de **projection** (quelques uns ou tous les champs peuvent être retenus dans le résultat) et l'opération de **sélection** (uniquement les enregistrements qui satisfont la condition sont prises en compte).

Pour afficher, par exemple, la liste des films de "**Sciences-fiction**". On peut taper la requête suivante :

```
SELECT *
FROM film
WHERE film_type = 'Science-fiction';
```

Le résultat de cette requête sera réduit aux enregistrements dont la colonne **film_type** vérifie la condition mentionnée : **film_type = 'Science-fiction'**.

film_id	film_titre	film_annee	film_type	film_pays
1	**Matrix**	**1999**	**Science-fiction**	**USA**
2	Fast and Furious	2001	Aventure	USA
3	John Wick	2014	Action	USA
4	**Matrix Reloaded**	**2003**	**Science-fiction**	**USA**

⇒

film_id	film_titre	film_annee	film_type	film_pays
1	Matrix	1999	Science-fiction	USA
4	Matrix Reloaded	2003	Science-fiction	USA

Autres Exemples :

Exemple 1 : Afficher le titre et l'année de lancement des films lancés après l'année 2000.

```
SELECT film_titre AS titre,
       film_annee AS année
FROM film
WHERE film_annee >= 2000;
```

Exemple 2 : Afficher le nom et la date de naissance des acteurs nées entre les années 1960 et 1969.

```
SELECT act_nom acteur,
       act_dn 'Date naissance'
FROM acteur
WHERE Year(act_dn) BETWEEN 1960 AND
1969;
```

Exemple 3 : Afficher le rôle et le numéro de film interprétés par l'acteur dont **`act_id`** est 1.

```
SELECT film_id, personnage
FROM acteur_film
WHERE act_id = 1;
```

Exemple 4 : Afficher les rôles joués par les acteurs qui contiennent le mot '**John**'.

```
SELECT personnage
FROM acteur_film
WHERE personnage LIKE '%John%';
```

4. Jointure

Une **jointure** est une opération permettant de combiner les enregistrements de deux ou de plusieurs tables en fonction de leurs colonnes communes.

Dans ce cas la requête de sélection doit mentionner toutes les tables desquelles on souhaite extraire les données ainsi que les tables qui les relient entre-elles. La syntaxe générale d'une jointure est comme suit :

```
SELECT [DISTINCT] * | colonne1, colonne2, …
FROM table1, table2, …
WHERE table1.pk = table2.fk AND
      condition;
```

Généralement, une jointure nécessite de relier la **clé primaire** de la table mère avec la **clé étrangère** de la table fille : **`table1.pk = table2.fk`**.

Dans cette syntaxe :

- "**`pk`**" symbolise le nom du champ utilisé comme **clé primaire** dans la table "**`table1`**".
- "**`fk`**" symbolise le nom du champ utilisé comme **clé étrangère** dans la table "**`table2`**".

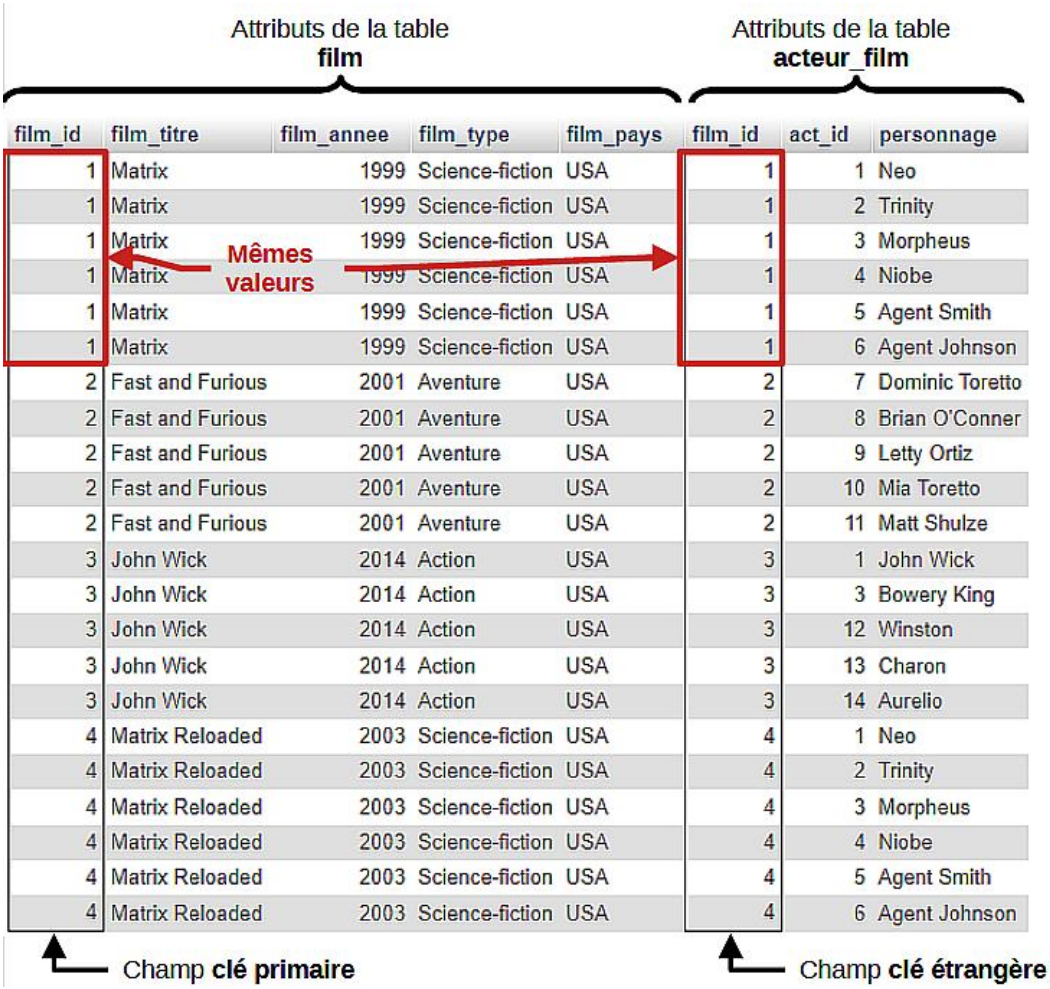
Pour simplifier l'écriture d'une requête SQL, effectuant des jointures entre plusieurs tables, on utilise souvent les **alias** de table pour raccourcir les noms de tables.

```
SELECT [DISTINCT] * | colonne1, colonne2, …
FROM table1 AS t1, table2 AS t2, …
WHERE t1.pk = t2.fk AND condition;
```

Pour comprendre le fonctionnement d'une jointure il suffit d'écrire la requête suivante :

```
SELECT *
FROM film AS f, acteur_film AS af
WHERE f.film_id = af.film_id;
```

Cette requête concerne tous les attributs de la table **film**, suivis par ceux de la table **acteur_film**. Pour chaque "**film_id**", clé primaire, de la table mère (**film**) on recherche tous les enregistrements de la table fille (**acteur_film**) dont le champ "**film_id**", clé étrangère possède la même valeur. Le résultat est présenté dans la figure 2.



film_id	film_titre	film_annee	film_type	film_pays	film_id	act_id	personnage
1	Matrix	1999	Science-fiction	USA	1	1	Neo
1	Matrix	1999	Science-fiction	USA	1	2	Trinity
1	Matrix	1999	Science-fiction	USA	1	3	Morpheus
1	Matrix	1999	Science-fiction	USA	1	4	Niobe
1	Matrix	1999	Science-fiction	USA	1	5	Agent Smith
1	Matrix	1999	Science-fiction	USA	1	6	Agent Johnson
2	Fast and Furious	2001	Aventure	USA	2	7	Dominic Toretto
2	Fast and Furious	2001	Aventure	USA	2	8	Brian O'Conner
2	Fast and Furious	2001	Aventure	USA	2	9	Letty Ortiz
2	Fast and Furious	2001	Aventure	USA	2	10	Mia Toretto
2	Fast and Furious	2001	Aventure	USA	2	11	Matt Shulze
3	John Wick	2014	Action	USA	3	1	John Wick
3	John Wick	2014	Action	USA	3	3	Bowery King
3	John Wick	2014	Action	USA	3	12	Winston
3	John Wick	2014	Action	USA	3	13	Charon
3	John Wick	2014	Action	USA	3	14	Aurelio
4	Matrix Reloaded	2003	Science-fiction	USA	4	1	Neo
4	Matrix Reloaded	2003	Science-fiction	USA	4	2	Trinity
4	Matrix Reloaded	2003	Science-fiction	USA	4	3	Morpheus
4	Matrix Reloaded	2003	Science-fiction	USA	4	4	Niobe
4	Matrix Reloaded	2003	Science-fiction	USA	4	5	Agent Smith
4	Matrix Reloaded	2003	Science-fiction	USA	4	6	Agent Johnson

*Figure 2: Résultat de la jointure entre les deux **tables film** et **acteur_film***

Pour sélectionner, seulement, les titres des films où on trouve le personnage '**Dominic Toretto**', on écrit :

```
SELECT film_titre
FROM film AS f, acteur_film AS af
WHERE f.film_id = af.film_id AND
  personnage = 'Dominic Toretto';
```

film_titre
Fast and Furious

Autres Exemples :

Exemple 1 : Afficher la liste de tous les acteurs du film **Matrix**.

```
SELECT act_nom 'Nom Acteur'
FROM film AS f,
     acteur AS a,
     acteur_film AS af
WHERE f.film_id = af.film_id AND
      a.act_id = af.act_id AND
      film_titre = 'Matrix';
```

Nom Acteur
Keanu Reeves
Carrie-Anne Moss
Laurence Fishburne
Jada Oinkett Smith
Hugo Weaving
Daniel Bernhardt

Exemple 2 : Afficher la liste des films (titre, année de lancement) dans lesquels l'acteur '**Keanu Reeves**' a participé.

```
SELECT film_titre AS titre,
       film_annee AS 'année lancement'
FROM film AS f, acteur AS a, acteur_film AS af
WHERE f.film_id = af.film_id AND
      a.act_id = af.act_id AND
      act_nom = 'Keanu Reeves';
```

titre	année lancement
Matrix	1999
John Wick	2014
Matrix Reloaded	2003

5. Agrégation

Les **requêtes d'agrégation** sont utilisées pour effectuer des calculs sur un ensemble de données dans une base de données relationnelle. Elles sont couramment utilisées pour extraire des informations agrégées, telles que la somme, la moyenne, le minimum, le maximum, ou le compte des valeurs dans une colonne d'une table.

Les **requêtes d'agrégation** sont particulièrement utiles pour générer des rapports statistiques.

Pour réaliser cette opération avec SQL, on utilise le mot clé **`GROUP BY`** suivi du nom du champ ou des champs sur lesquels s'effectue l'agrégat.

```
SELECT [DISTINCT] select_expr₁[, select_expr₂ ...]
FROM références_tables
[WHERE condition]
[GROUP BY {nom_col | position} [ASC | DESC], ...;
```

Avec l'instruction **`GROUP BY`**, on peut exploiter les **fonctions d'agrégation** dans les "**`select_exprᵢ`**" devant la clause **`SELECT`**.

Les **fonctions d'agrégation** permettent d'effectuer diverses opérations statistiques sur des ensembles de valeurs. Parmi ces fonctions, on peut citer :

- **`SUM()`** : pour calculer la somme des valeurs d'un ensemble,
- **`AVG()`** : pour calculer la valeur moyenne d'un ensemble,
- **`COUNT()`** : pour compter le nombre d'enregistrement dans un ensemble,
- **`MAX()`** : pour récupérer la valeur maximale d'un ensemble. Cette fonction s'applique à la fois aux données numériques ou alphanumériques.
- **`MIN()`** :pour récupérer la valeur minimum de la même manière que **`MAX()`**.

On veut afficher le "**film_id**" et le nombre d'acteurs (**NombreActeurs**) dans chaque film. La requête SQL s'écrit comme suit :

```
SELECT film_id,
       COUNT(*) AS 'NombreActeurs'
FROM acteur_film
GROUP BY film_id;
```

film_id	NombreActeurs
1	6
2	5
3	5
4	6

La figure 3 montre le principe de fonctionnement d'une requête d'agrégation. Les données de la colonne "**`film_id`**" sont, initialement, regroupés selon les valeurs de ce champ. La fonction **`COUNT(*)`** est une **fonction d'agrégation** (ou d'agrégat) qui compte le nombre de lignes résultantes dans chaque ensemble.

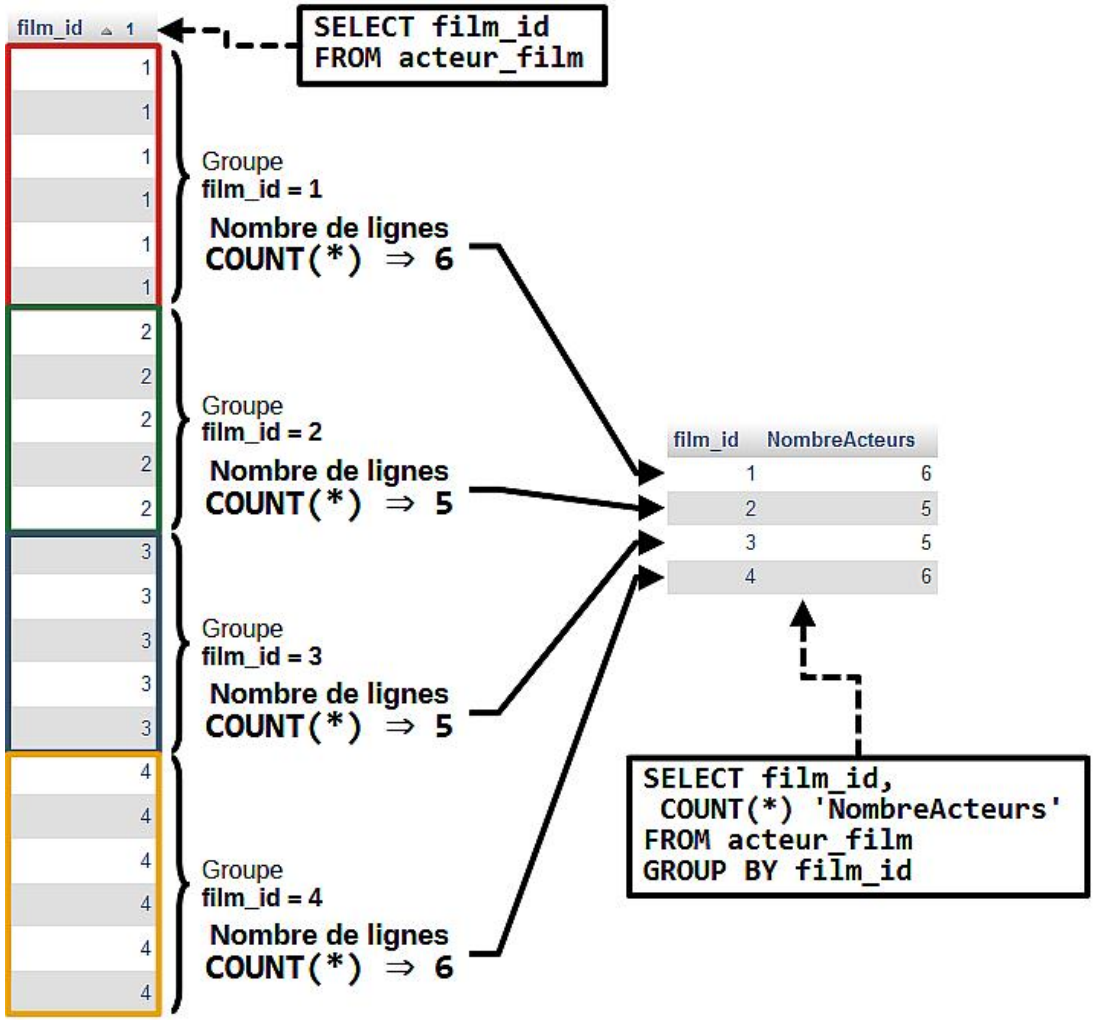


Figure 3: Nombre d'acteurs dans un film donné

Au lieu d'indiquer le nom du champ (**film_id**) devant le mot clé **GROUP BY**, on peut utiliser la position de l'élément de groupage dans la liste des colonnes devant la clause **SELECT** (ici 1 indique que le groupement se fait selon la première colonne).

```
SELECT film_id, COUNT(*) AS 'NombreActeurs'
FROM acteur_film
GROUP BY 1;
```

film_id	NombreActeurs
1	6
2	5
3	5
4	6

Autres Exemples :

Exemple 1 : Regrouper les acteurs par leurs mois de naissance tout en affichant le nombre d'acteurs né chaque mois.

```
SELECT MONTH(act_dn) AS Mois,
  COUNT(*) AS 'NbActeurs'
FROM acteur
GROUP BY MONTH(act_dn);
```

```
SELECT MONTH(act_dn) AS Mois,
  COUNT(*) AS 'NbActeurs'
FROM acteur
GROUP BY 1;
```

Mois	NbActeurs
4	2
7	6
8	2
9	4

MONTH(act_dn) retourne le mois de naissance d'un acteur.
Devant la clause **GROUP BY** il est possible d'utiliser soit l'expression (**MONTH(act_dn)**), soit la position de l'expression dans la clause **SELECT** (ici **1**), ou bien l'alias (**Mois**).

Exemple 2 : Afficher le nombre d'acteurs différents qui ont participé dans chaque type de film.
La requête suivante est erronée. En effet, les deux films "**Matrix**" et "**Matrix Reloaded**" sont de type "**Science-fiction**" et ils ont été joués par les mêmes six acteurs.

```
SELECT film_type 'TypeFilm', COUNT(act_id) 'NbActeurs'
FROM film AS f, acteur_film AS af
WHERE f.film_id = af.film_id
GROUP BY film_type;
```

TypeFilm	NbActeurs
Action	5
Aventure	5
Science-fiction	12

Pour corriger l'erreur, et supprimer les duplications du champ **act_id** on doit utiliser **DISTINCT act_id** dans la requête.

```
SELECT film_type AS 'TypeFilm',
  COUNT(DISTINCT act_id) AS 'NbActeurs'
FROM film AS f, acteur_film AS af
WHERE f.film_id = af.film_id
GROUP BY film_type;
```

TypeFilm	NbActeurs
Action	5
Aventure	5
Science-fiction	6

Exemple 3 : Déterminer la date de naissance de l'acteur le plus vieux de chaque film.

```
SELECT f.film_id, f.film_titre,
       MIN(act_dn) AS 'DateNaiss'
FROM acteur_film AS af, acteur AS a, film AS f
WHERE af.act_id = a.act_id AND
      af.film_id = f.film_id
GROUP BY f.film_id, f.film_titre;
```

film_id	film_titre	DateNaiss
1	Matrix	1960-04-04
2	Fast and Furious	1967-07-18
3	John Wick	1942-09-29
4	Matrix Reloaded	1960-04-04

Nous devons mentionner la table "**acteur_film**" dans la clause **FROM** car elle relie les deux tables "**film**" et "**acteur**".

Pour afficher les titre des films comportant exactement 6 acteurs. On pourra penser que la requête suivante, fait l'affaire :

```
SELECT film_titre,
       COUNT(*) AS 'NombreActeurs'
FROM acteur_film AS af, film AS f
WHERE af.film_id = f.film_id AND
      COUNT(*) = 6
GROUP BY 1;
```

Au contraire, elle provoque une erreur car les fonctions d'agrégat (**COUNT(*) = 6**) ne sont pas autorisés dans la clause **WHERE**.

Erreur

Requête SQL : Copier

```
SELECT film_titre,
       COUNT(*) AS 'NombreActeurs'
FROM acteur_film AS af, film AS f
WHERE af.film_id = f.film_id AND
      COUNT(*) = 6
GROUP BY 1 LIMIT 0, 25
```

MySQL a répondu :

```
#1111 - Utilisation invalide de la clause GROUP
```

Pour corriger l'erreur, il faut utiliser une clause **HAVING** après le **GROUP BY**. Cette clause agit exactement comme la clause **WHERE**, mais elle est évaluée après l'opération de regroupement (**GROUP BY**).

La requête finale est la suivante :

```
SELECT film_titre,
       COUNT(*) AS `Nombre Acteurs`
FROM acteur_film AS af, film AS f
WHERE af.film_id = f.film_id
GROUP BY film_titre
HAVING COUNT(*) = 6;
```

Utilisation de la fonction d'**agrégat**

```
SELECT film_titre,
       COUNT(*) AS `Nombre Acteurs`
FROM acteur_film AS af, film AS f
WHERE af.film_id = f.film_id
GROUP BY film_titre
HAVING `Nombre Acteurs` = 6;
```

Utilisation de l'**alias**

Son résultat est présenté ci-après.

film_titre	Nombre Acteurs
Matrix	6
Matrix Reloaded	6

Autres Exemples :

Exemple : Déterminer la liste des acteurs (id, nom) qui ont participé à un seul film.

```
SELECT a.act_id, act_nom, COUNT(*) AS `NbreFilms`
FROM acteur AS a, acteur_film AS af
WHERE a.act_id = af.act_id
GROUP BY a.act_id, act_nom
HAVING `NbreFilms` = 1;
```

act_id	act_nom	NbreFilms
7	Vin Diesel	1
8	Paul Walker	1
9	Michelle Rodriguez	1
10	Jordana Brewster	1
11	Matt Schulze	1
12	Ian Mc Shane	1
13	Lance Reddick	1
14	John Leguizamo	1

6. Sous-requêtes

Les **sous-requêtes**, également appelées **requêtes imbriquées**, en SQL sont des requêtes incorporées dans d'autres requêtes. Elles sont utilisées pour récupérer des données plus complexes en combinant les résultats de plusieurs requêtes.

Bien qu'une **sous-requête** soit fréquemment insérée devant la clause **`WHERE`** d'une requête, comme le montre la figure 4, on peut aussi la retrouver dans d'autres emplacements dans la requête.

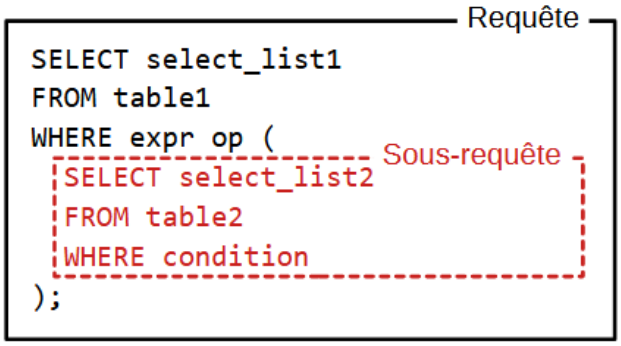


Figure 4: Requêtes imbriquées / Sous-requêtes

La requête présentée en figure 4 est évaluée selon le séquencement suivant :

- La **sous-requête** est évalué en premier,
- Le résultat de la **sous-requête** est, par la suite, exploité pour exécuter la **requête principale**.

Pour afficher l'acteur le plus vieux (**Ian Mc Shane**, né le **29/09/1942**), nous pensons immédiatement à la requête suivante, dont le résultat est présenté ci-contre :

```
SELECT act_nom, MIN(act_dn) AS `Date Naissance`
FROM acteur;
```

act_nom	Date Naissance
Keanu Reeves	1942-09-29

Bien que la date de naissance soit correcte, le nom d'acteur lui il n'est pas correct. Nous nous attendons que le nom d'acteur soit : **Ian Mc Shane**.

On devine très rapidement qu'il faudra recourir aux **sous-requêtes** afin de corriger cette requête. En effet, il faut :

- Premièrement, retrouver la date de naissance minimale dans la table "**acteur**".
- Ensuite, rechercher les acteurs qui y sont nés.

Pour afficher les acteurs les plus vieux, la **sous-requête recherche la date de naissance minimale des acteurs**, puis le résultat est exploité pour retrouver tous les acteurs qui sont nés à cette date là.

```
SELECT *
FROM acteur
WHERE act_dn = (
  /* sous requête */
  SELECT MIN(act_dn) FROM acteur
);
```

act_id	act_nom	act_dn
12	Ian Mc Shane	1942-09-29

Également, pour afficher les noms des acteurs du film "**Matrix**" on peut utiliser une **sous-requête**.

La **sous-requête** détermine **les identifiants (`act_id`) des acteurs qui ont participé dans le film**. La requête (externe) trouve tous les acteurs ayant les (**`act_id`**) trouvés (dans la sous-requête).

```
SELECT * FROM acteur
WHERE act_id IN (
  SELECT act_id
  FROM film AS f, acteur_film AS af
  WHERE f.film_id = af.film_id AND
    film_titre = 'Matrix');
```

act_id	act_nom	act_dn
1	Keanu Reeves	1964-09-02
2	Carrie-Anne Moss	1967-08-21
3	Laurence Fishburne	1961-07-30
4	Jada Oinkett Smith	1971-09-18
5	Hugo Weaving	1960-04-04
6	Daniel Bernhardt	1965-08-31

Autres Exemples :

Exemple 1 : Afficher les noms des acteurs de "**Matrix**" (titre du film et nom de l'acteur) qui ont participé dans d'autres films.

La sous-requête cherche, dans la table "**acteur_film**", les identifiants des acteurs (**`act_id`**) qui ont participé au film "**Matrix**". Le résultat de la sous-requête est employé pour retrouver les noms des acteurs et des films autres que "**Matrix**" auxquels ils ont participé.

```
SELECT film_titre, act_nom
FROM acteur AS a,
     film AS f,
     acteur_film AS af
WHERE a.act_id = af.act_id AND
  f.film_id = af.film_id AND
  film_titre <> 'Matrix' AND
  a.act_id IN (
    SELECT act_id
    FROM film AS f1,
         acteur_film AS af1
    WHERE f1.film_id = af1.film_id AND
      film_titre = 'Matrix'
  )
ORDER BY act_nom, film_titre;
```

film_titre ▲ 2	act_nom ▲ 1
Matrix Reloaded	Carrie-Anne Moss
Matrix Reloaded	Daniel Bernhardt
Matrix Reloaded	Hugo Weaving
Matrix Reloaded	Jada Oinkett Smith
John Wick	Keanu Reeves
Matrix Reloaded	Keanu Reeves
John Wick	Laurence Fishburne
Matrix Reloaded	Laurence Fishburne

7. Mise en ordre

Après la sélection des données il est fréquemment nécessaire d'ordonner (ou de trier) le résultat selon les valeurs d'une ou de plusieurs colonnes. L'opération est possible grâce à la clause **`ORDER BY`**. Cette clause doit impérativement être placée après les clauses **`GROUP BY`** et **`HAVING`**, comme suit :

```
SELECT [ALL | DISTINCT] select_expr [, select_expr] ...
[FROM table_references]
[WHERE where_condition]
[GROUP BY {col_name | expr | position}, ...]
[HAVING where_condition]
[ORDER BY {col_name | expr | position} [ASC | DESC], ...]
```

Le tri peut non seulement se faire selon le nom, ou l'alias, d'une colonne **`col_name`** de la clause **`SELECT`**, mais aussi selon la valeur d'une expression **`expr`**, ou selon la position d'une colonne dans la clause **`SELECT`**.

Le tri d'une colonne peut se faire en ordre croissant ↗ "**`ASC`**", ou en ordre décroissant ↘ "**`DESC`**". Par défaut, lorsque l'ordre n'est pas signalé dans la requête, le tri est effectué en ordre croissant "**`ASC`**".

Exemples :

Exemple 1 : Afficher tous les acteurs ordonnés alphabétiquement par leurs noms.

```
SELECT * FROM acteur
ORDER BY act_nom;
```

act_id	act_nom ▲ 1	act_dn
2	Carrie-Anne Moss	1967-08-21
6	Daniel Bernhardt	1965-08-31
5	Hugo Weaving	1960-04-04
12	Ian Mc Shane	1942-09-29
4	Jada Oinkett Smith	1971-09-18
14	John Leguizamo	1960-07-22
10	Jordana Brewster	1980-04-26
1	Keanu Reeves	1964-09-02
13	Lance Reddick	1962-07-07
3	Laurence Fishburne	1961-07-30
11	Matt Schulze	1972-07-03
9	Michelle Rodriguez	1978-07-12
8	Paul Walker	1973-09-12
7	Vin Diesel	1967-07-18

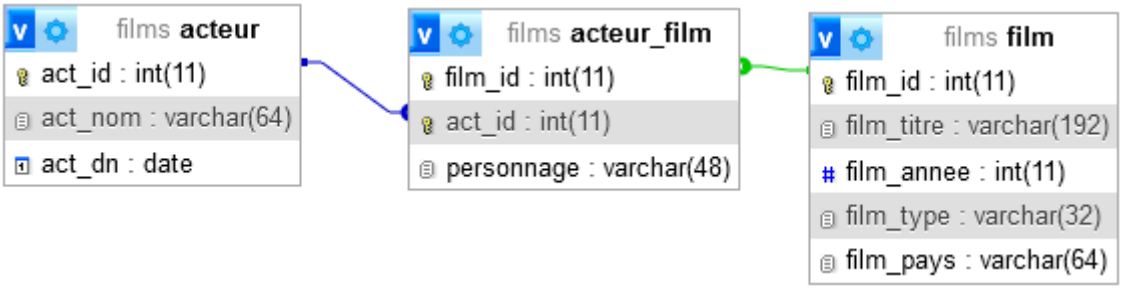
Exemple 2 : Afficher tous les acteurs ordonnés du plus jeune au plus vieux selon leurs dates de naissance.

```
SELECT * FROM acteur
ORDER BY act_dn DESC;
```

act_id	act_nom	act_dn ▼ 1
10	Jordana Brewster	1980-04-26
9	Michelle Rodriguez	1978-07-12
8	Paul Walker	1973-09-12
11	Matt Schulze	1972-07-03
4	Jada Oinkett Smith	1971-09-18
2	Carrie-Anne Moss	1967-08-21
7	Vin Diesel	1967-07-18
6	Daniel Bernhardt	1965-08-31
1	Keanu Reeves	1964-09-02
13	Lance Reddick	1962-07-07
3	Laurence Fishburne	1961-07-30
14	John Leguizamo	1960-07-22
5	Hugo Weaving	1960-04-04
12	Ian Mc Shane	1942-09-29

Annexe 1 : Base de données films

Représentation graphique de la base de données **films**



Données de la base de données **films**

Table **film**

film_id	film_titre	film_annee	film_type	film_pays
1	Matrix	1999	Science-fiction	USA
2	Fast and Furious	2001	Aventure	USA
3	John Wick	2014	Action	USA
4	Matrix Reloaded	2003	Science-fiction	USA

Table **acteur**

act_id	act_nom	act_dn
1	Keanu Reeves	1964-09-02
2	Carrie-Anne Moss	1967-08-21
3	Laurence Fishburne	1961-07-30
4	Jada Oinkett Smith	1971-09-18
5	Hugo Weaving	1960-04-04
6	Daniel Bernhardt	1965-08-31
7	Vin Diesel	1967-07-18
8	Paul Walker	1973-09-12
9	Michelle Rodriguez	1978-07-12
10	Jordana Brewster	1980-04-26
11	Matt Schulze	1972-07-03
12	Ian Mc Shane	1942-09-29
13	Lance Reddick	1962-07-07
14	John Leguizamo	1960-07-22

Table **acteur_film**

film_id	act_id	personnage
1	1	Neo
1	2	Trinity
1	3	Morpheus
1	4	Niobe
1	5	Agent Smith
1	6	Agent Johnson
2	7	Dominic Toretto
2	8	Brian O'Conner
2	9	Letty Ortiz
2	10	Mia Toretto
2	11	Matt Shulze
3	1	John Wick
3	3	Bowery King
3	12	Winston
3	13	Charon
3	14	Aurelio
4	1	Neo
4	2	Trinity
4	3	Morpheus
4	4	Niobe
4	5	Agent Smith
4	6	Agent Johnson

Annexe 2 : Fonctions usuelles

1. Opérateurs

Le tableau suivant présente les opérateurs usuels qui peuvent être employés dans une expression booléenne :

Type	Opérateur	Exemple

Comparaison	`=, >, <, >=, <= et <>`	`-- Liste des films parus après l'année 2011`
		`SELECT * FROM film`
		`WHERE film_annee >= 2011;`

Intervalle	`BETWEEN … AND …`	`-- Liste les acteurs nés aux années 70`
		`SELECT * FROM acteur`
		`WHERE act_dn BETWEEN '1970-01-01' AND`
		`'1979-12-31';`

Listes	`champ IN (v1, …)`
	`champ NOT IN (v1, …)`
		`-- Liste des acteurs ayant les act_id spécifiés`
		`SELECT * FROM acteur`
		`WHERE act_id IN (3, 6, 12);`

Valeurs nulles	`IS NULL`
	`IS NOT NULL`
		`-- Liste des acteurs dont la date de naissance`
		`-- est vide`
		`SELECT * FROM acteur`
		`WHERE act_dn IS NULL;`

		`-- Liste des films`
		`-- dont film_type n'est pas vide`
		`SELECT * FROM film`
		`WHERE film_type IS NOT NULL;`

Ressemblance	`champ LIKE motif`	`-- Liste des personnages commençant par 'M'`
		`-- '%' remplace de 0 à n caractères`
		`SELECT DISTINCT personnage FROM acteur_film`
		`WHERE personnage LIKE 'M%';`

		`-- Liste des films produits entre 2010 et 2019`
		`-- '_' remplace un seul caractère`
		`SELECT * FROM film`
		`WHERE film_annee LIKE '201_';`

Logiques	`NOT expression`
	`expr1 AND expr2`
	`expr1 OR expr2`
		`-- Liste des films`
		`-- qui ne sont pas des films de 'Science-fiction'`
		`-- qui sont produits après l'année 2000`
		`SELECT * FROM film`
		`WHERE film_annee > 2000 AND`
		`film_titre <> 'Science-fiction';`

		`-- Liste des films produits en 2001 et en 2014`
		`SELECT * FROM film`
		`WHERE film_annee = 2001 OR film_annee = 2014;`

2. Fonctions sur les dates

Fonction	Description	Exemple

`YEAR(date)`
`MONTH(date)`
`DAY(date)`
	Extraire respectivement
	l'année,
	le numéro du mois et
	le jour du mois
	depuis date
		`-- Afficher l'année, le mois, et`
		`-- le jour de naissance des acteurs`
		`SELECT act_dn AS `Date`,`
		`  YEAR(act_dn) AS `Année`,`
		`  MONTH(act_dn) AS `Mois`,`
		`  DAY(act_dn) AS `jour``
		`FROM acteur;`

`NOW()`	Retourne la date système	`-- Afficher la date système` `SELECT NOW();`