

# TD 3 : SQL

## SQL Interrogation de données, requêtes complexes

2025-10-10

---

L3 MIASHS  
[Université Paris Cité](#)  
Année 2025  
[Course Homepage](#)  
[Moodle](#)




### Objectifs

- requêtes imbriquées,
- jointures externes,
- vues,
- fonctions SQL.

### Prérequis

- schéma [world](#),
- schéma [pagila](#),
- cours [Algèbre relationnelle](#),
- cours [Requêtes simples](#),
- cours [Requêtes imbriquées, CTEs](#).


-  Quand on travaille sur plusieurs schémas (ici `world`, `pagila` et votre schéma personnel), il est bon
  - d'ajuster `search_path` : `set search_path to world, pagila, ...` ; (remplacer `...` par votre identifiant)
  - de qualifier les noms de table pour indiquer le schéma d'origine : `world.country` versus `pagila.country`

### Questions

1. Quels sont les langues qui ne sont officielles dans aucun pays ? (355 lignes)

Écrivez une version avec `EXCEPT`, une avec `NOT IN` et une autre avec `LEFT JOIN`.

2. Quelles sont les régions où au moins deux pays ont la même forme de gouvernement ? (21 lignes)
3. Quels sont les films qui n'ont jamais été loués ? (42 lignes)

-  Cette question est exactement du même type que la précédente. On y répond de la même manière : pour trouver les objets d'un certain type qui ne possèdent pas une propriété, on cherche dans la base tous les objets de ce type et on fait la différence avec l'ensemble des objets de ce type qui possèdent la propriété dans la base.

4. Quels sont les acteurs qui ont joué dans toutes les catégories de film ? (11 lignes)

! Cette requête réalise une opération sophistiquée de l'algèbre relationnelle la *division* ou  $\div$ . Il ne s'agit pas d'une opération primitive comme  $\sigma$ ,  $\pi$ ,  $\times$ .

$$\pi_{\text{actor\_id,category\_id}}(\text{film\_actor} \bowtie \text{film\_category}) \div \pi_{\text{category}}(\text{film\_category})$$

5. Existe-t-il des acteurs qui ne jouent avec aucun autre acteur ? (0 ligne)
6. Nom, prénom des clients installés dans des villes sans magasin ? (599 lignes)
7. Lister les pays pour lesquels toutes les villes ont au moins un magasin. (1 ligne)
8. Déterminer la liste des films disponibles dans toutes les langues.
9. Un même *dvd* (*inventory\_id*) peut bien sûr être loué plusieurs fois, mais pas simultanément. Proposer une requête qui vérifie que les dates de location d'un *dvd* donné sont compatibles.

## Vues

Les *vues* permettent de donner un nom à une requête afin de pouvoir l'appeler plus tard sans la réécrire à chaque fois. Une vue s'enregistre dans un schéma. Par exemple, dans le schéma **World**, on pourrait créer une vue **VillesRepublic** qui contient toutes les villes de la table *city* qui sont dans une république.

On crée une vue avec **CREATE VIEW nom AS requete**. Étant donné que vous ne pouvez écrire que dans votre schéma personnel, il faudra nommer vos vues **entid.nom** où **entid** est votre identifiant ENT. Ainsi

```
CREATE VIEW entid.VillesRepublic AS
SELECT
  B.*
FROM
  world.country as A
NATURAL JOIN
  world.city as B
WHERE
  A.governmentform like '%Republic%';
```

créer une vue dans votre schéma personnel. Désormais, si on veut sélectionner les villes qui sont dans une république et dont la population est supérieure à 1000000, on pourra simplement écrire :

```
SELECT *
FROM
  entid.VillesRepublic
WHERE
  population >= 1000000;
```

i Remarquez la différence entre **WITH** et une vue. **WITH** nomme une requête temporairement, seulement à l'échelle de la requête courante tandis qu'une vue est enregistrée de façon permanente. Cependant, chaque fois que vous appelez votre vue, elle est réévaluée par le système de base de données.

Notez aussi que SQL n'est pas sensible à la casse. La vue **entid.VillesRepublic** peut être aussi désignée par **entid.villesrepublic**.

Pour supprimer une vue existante on utilise la commande **DROP VIEW** suivie du nom de la vue à supprimer. Par exemple l'instruction

```
DROP VIEW entid.VillesRepublic ;
```

supprime la vue créée précédemment.

Dans votre schéma personnel (qui porte le nom de votre identifiant ENT), écrire une vue `film_id_horror` qui renvoie la liste des films de catégorie 'Horror'.

## Fonctions SQL

Dans votre schéma personnel (qui porte le nom de votre identifiant ENT), écrire une fonction SQL `film_id_cat` qui prend en paramètre une chaîne de caractère `s` et renvoie la liste des films de catégorie `s`. On rappelle la syntaxe :

```
CREATE OR REPLACE FUNCTION entid.film_id_cat(s TEXT)
RETURNS TABLE(film_id INTEGER)
LANGUAGE 'sql' AS
$$
requete
$$
```

et l'usage

```
CREATE OR REPLACE FUNCTION
    entid.film_id_cat(s text)
RETURNS TABLE(film_id smallint) AS
$$
    SELECT fc.film_id
    FROM
        pagila.film_category fc
    JOIN
        pagila.category ca
    ON (fc.category_id=ca.category_id)
    WHERE
        ca.name=s ;
$$ LANGUAGE sql ;
```

## Questions

Utilisez votre fonction pour écrire les requêtes suivantes :

10. Quels sont les acteurs qui ont déjà joué dans un film d'horreur (catégorie 'Horror') ?
11. Quels sont les acteurs qui n'ont jamais joué dans une comédie (Comedy) ? (53 lignes)

🔥 ⚠ Attention ! Cette requête ne répond pas à la question :

```
SELECT DISTINCT ac.*
FROM pagila.actor ac NATURAL JOIN
    (SELECT * FROM pagila.film_actor
     WHERE film_id NOT IN
        (SELECT * FROM pagila.film_id_cat('Comedy') )
    ) as X;
```

Elle répond à la question : *Quels sont les acteurs qui ont joué dans un film qui n'est pas une comédie ?*

**i** Nous pouvons par exemple d'abord calculer la liste des `actor_id` des acteurs qui ont joué dans au moins une comédie, puis calculer la liste des `actor_id` des acteurs présents dans la base et faire la différence

12. Quels sont les acteurs qui ont joué dans un film d'horreur ('Horror') et dans un film pour enfant ('Children') ? (130 lignes)

🔥 Ici l'erreur la plus fréquente consiste à écrire

```
SELECT
  actor_id
FROM
  pagila.film_actor AS fa
WHERE
  fa.film_id IN (
    SELECT *
    FROM entid.film_id_cat('Children')
  ) AND
  fa.film_id IN (
    SELECT *
    FROM entid.film_id_cat('Horror')
  );
```

Le résultat est vide et la requête ne correspond pas à la question posée.

Elle calcule les **actor\_id** des acteurs qui ont dans au moins un film qui relève simultanément des catégories **Horror** et **Children** (ce genre de film est assez rare).

Pour calculer un résultat correct, il faut pour chaque valeur  $a$  de **actor\_id** rechercher deux tuples (pas nécessairement distincts) de **film\_actor**, où l'attribut **actor\_id** vaut  $a$ , et tel que, dans un cas, **film\_id** désigne un film pour enfants et, dans l'autre, un film d'horreur. En algèbre relationnelle, cela donne :

$$\pi_{\text{last\_name}, \text{first\_name}} \left( \text{actor} \bowtie \left( \left( \pi_{\text{actor\_id}} (\text{film\_actor} \bowtie \text{film\_id\_cat}('Children')) \right) \cap \left( \pi_{\text{actor\_id}} (\text{film\_actor} \bowtie \text{film\_id\_cat}('Horror')) \right) \right) \right)$$