TD 3: SQL

SQL Interrogation de données, requêtes complexes

2025-10-10



Avec solutions

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
 - 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.

```
Solution

(
    SELECT DISTINCT language
    FROM world.countrylanguage
)

EXCEPT

(
    SELECT language
    FROM world.countrylanguage
    WHERE isofficial
);
```

Première version

```
Solution

SELECT DISTINCT language
FROM world.countrylanguage
WHERE language NOT IN

(SELECT language
FROM world.countrylanguage
WHERE isofficial);
```

Deuxième version:

```
Solution

SELECT DISTINCT 11.language
FROM world.countrylanguage AS 11
    LEFT JOIN world.countrylanguage AS 1
    ON (11.language = 1.language AND 1.isofficial)
WHERE 1.language IS NULL;
```

Troisième version:

```
SELECT DISTINCT cl.language
FROM world.countrylanguage cl
WHERE NOT EXISTS (
SELECT cl1.language
FROM world.countrylanguage cl1
WHERE cl1.language=cl.language AND
cl1.isofficial
);
```

2. Quelles sont les régions où au moins deux pays ont la même forme de gouvernement? (21 lignes)

```
SELECT DISTINCT region
FROM world.country AS c1
WHERE c1.governmentform = ANY(
    SELECT c2.governmentform
    FROM world.country AS c2
    WHERE c2.countrycode!=c1.countrycode AND c2.region=c1.region
);
```

Solution

```
SELECT DISTINCT c1.region

FROM world.country AS c1 JOIN world.country AS c2

ON c1.region=c2.region AND

c1.countrycode!=c2.countrycode AND

c1.governmentform=c2.governmentform;
```

3. Quels sont les films qui n'ont jamais été loués? (42 lignes)

Solution

Là encore, plusieurs possibilités. Avec ce que l'on sait déjà :

```
WITH DejaLoue AS (
SELECT film_id
FROM rental JOIN inventory USING (inventory_id)
), NonLoue AS (
SELECT film_id
FROM film
EXCEPT
SELECT *
FROM DejaLoue
)

SELECT title
FROM film NATURAL JOIN NonLoue;

Avec les requêtes imbriquées:

SELECT title,film_id FROM film
WHERE film id NOT IN (
```

SELECT title,film_id FROM film

WHERE film_id NOT IN (

SELECT film_id

FROM rental JOIN inventory USING (inventory_id)
);

- **i** 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)

```
WITH ActCat AS (SELECT actor_id, category_id FROM film_actor fa

JOIN film_category fc ON (fa.film_id=fc.film_id)),

ActNot AS (SELECT actor_id FROM actor,category

WHERE (actor_id,category_id) NOT IN (SELECT * FROM ActCat)),

ActId AS (SELECT actor_id FROM actor

EXCEPT SELECT * FROM ActNot)

SELECT first_name,last_name FROM actor NATURAL JOIN ActId ;
```

Cette requête réalise une opération sophistiquée de l'algèbre relationnelle la division ou \div . Il ne s'agit pas d'une opération primitive comme σ, π, \times .

 $\pi_{\texttt{actor_id}, \texttt{category_id}} \left(\texttt{film_actor} \bowtie \texttt{film_category} \right) \div \pi_{\texttt{category}} \left(\texttt{film_category} \right)$

Solution

La version suivante calcule le même résultat, et suit fidèlement le plan d'exécution le plus élémentaire pour réaliser la division.

```
WITH
  ActCat AS (
    SELECT actor_id, category_id
    FROM film_actor fa JOIN film_category fc ON (fa.film_id=fc.film_id)),
  ActCrosCat AS (
    SELECT actor_id, category_id
    FROM actor, category),
  ActNotCat AS (
    SELECT *
    FROM ActCrosCat
    EXCEPT
    SELECT *
    FROM ActCat),
  ActId AS (
    SELECT actor_id
    FROM actor
    EXCEPT
    SELECT actor_id
    FROM ActNotCat)
SELECT first_name, last_name
FROM actor NATURAL JOIN ActId ;
```

En comptant le nombre n de catégories de films dans une première requête, on peut aussi sélectionner les acteurs qui apparaissent dans au moins n catégories de film.

5. Existe-t-il des acteurs qui ne jouent avec aucun autre acteur? (0 ligne)

Solution Avec une jointure externe WITH copain AS (SELECT DISTINCT R1.actor_id FROM film_actor as R1 JOIN film_actor as R2 ON (R1.film_id = R2.film_id AND R1.actor_id != R2.actor_id)) SELECT actor_id FROM actor LEFT JOIN copain USING(actor_id) WHERE actor_id IS NULL; Avec une sous-requête ou une CTE : WITH copain AS (SELECT R1.actor_id FROM film_actor as R1 JOIN film_actor as R2 ON (R1.film_id = R2.film_id AND R1.actor_id != R2.actor_id)) SELECT actor_id FROM actor WHERE actor_id NOT IN (SELECT * FROM copain); ou avec NOT EXISTS SELECT actor_id FROM actor a WHERE NOT EXISTS (SELECT fa2.actor_id FROM film_actor fa1 film_actor fa2 ON (fa1.actor_id=a.actor_id AND fa2.actor_id<> a.actor_id AND fa1.film_id=fa2.film_id))

ChatGPT fecit (ça marche mais ça n'est pas très beau)

```
WITH ActorFilmCounts AS (
    -- Pour chaque acteur, chaque film, nombre de co-acteurs dans le film
    SELECT
        fa1.actor_id,
        fa1.film_id,
        COUNT(fa2.actor_id) AS other_actors_count
        film actor fa1
    LEFT JOIN
        film_actor fa2
        fa1.film_id = fa2.film_id
        AND fa1.actor_id <> fa2.actor_id
    GROUP BY
        fa1.actor_id, fa1.film_id
)
SELECT
    a.actor_id,
    a.first_name,
    a.last_name
FROM
    ActorFilmCounts afc
JOIN
    actor a
ON
    afc.actor_id = a.actor_id
GROUP BY
    a.actor_id, a.first_name, a.last_name
HAVING
    -- garder les acteurs qui n'ont pas de co-acteurs
    SUM(afc.other_actors_count) = 0;
```

∆ Une erreur

🛣 La requête suivante ne répond pas à la question posée :

```
SELECT
   fa.actor_id
FROM
   film_actor fa
LEFT JOIN
   film_actor fa2
ON
   (fa.film_id = fa2.film_id AND fa.actor_id<>fa2.actor_id)
WHERE
   fa2.actor_id IS NULL;
```

Elle liste les identifiants d'acteurs qui ont joué au moins une fois dans un film ne comportant qu'un seul acteur, alors qu'on ne cherche les acteurs qui n'ont joué que dans des films ne comportant qu'un seul acteur.

Cette requête renvoie 31 lignes.

6. Nom, prénom des clients installés dans des villes sans magasin? (599 lignes)

Avec une jointure externe :

```
WITH
  CustomerCity AS (
    SELECT
      cu.first_name,
      cu.last_name,
      cu.customer_id,
      ad.city_id
    FROM
      customer cu
    JOIN
      address ad
      (cu.address_id=ad.address_id)),
  StoreCity AS (
    SELECT
      ad.city_id
    FROM
      store st
    JOIN address ad
      (st.address_id= ad.address_id)
)
SELECT
  first_name,
  last_name
FROM
  CustomerCity cc LEFT JOIN StoreCity sc USING(city_id)
  sc.city_id IS null;
```

Avec une requête imbriquée :

```
WITH
  CustomerCity AS (
    SELECT
      cu.first_name,
      cu.last_name,
      cu.customer_id,
      ad.city_id
    FROM
      customer cu
    JOIN
      address ad
    ON
      (cu.address_id=ad.address_id)),
  StoreCity AS (
    SELECT
      ad.city_id
    FROM
      store st
    JOIN address ad
      (st.address_id= ad.address_id)
```

8

) SELECT

first name.

7. Lister les pays pour lesquels toutes les villes ont au moins un magasin. (1 ligne)

```
Solution
Sans requête imbriquée, seulement avec des jointures :
with co_with_ci_without_store as
    select distinct *
    from country co join city ci using (country_id)
        left join (address join store using (address_id)) ad_st
            using (city_id)
    where ad_st.store_id is null
)
select co.country
from country co left join co_with_ci_without_store cw on co.country_id = cw.country_id
where cw.country_id is null;
Avec des requêtes imbriquées
SELECT co.country
from country co
WHERE NOT EXISTS (
  SELECT *
  FROM city ci
  WHERE co.country_id=ci.country_id AND ci.city_id NOT IN
       SELECT address.city_id
       FROM store JOIN address USING (address_id)
    )
);
```

8. Déterminer la liste des films disponibles dans toutes les langues.

Solution

Comme pour les acteurs "toutes catégories", il s'agit d'une division. Dans la base installée, le résultat est vide.

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.

Solution

SQL en général et PostGres en particulier proposent beaucoup de types et d'opérations sophistiquées pour représenter et manipuler les données temporelles. Plusieurs types de données permettent de représenter les instants (timestamp), les dates, les intervalles de temps, les durées, et de calculer sur le temps (ajouter une durée à une date, extraire une information calendaire d'une date ou d'un instant, …). Même si l'API varie d'un cadre à l'autre, on retrouve ces types et ces opérations dans tous les environnements de sciences des données : \P , \P

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 en salle de TP. Ainsi

```
CREATE VIEW entid.VillesRepublic AS

SELECT

B.*

FROM

world.country as A

NATURAL JOIN

world.city as B

WHERE

A.governmentform like '%Republic%';
```

crée 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;
```

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. Villes Republic peut être aussi désignée par entid. villes republic.

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. Villes Republic ;
```

supprime la vue créée précédemment.

Dans votre schéma personnel entid (il faut remplacer entid par votre nom de login), écrire une vue film_id_horror qui renvoie la liste des films de catégorie 'Horror'.

```
CREATE VIEW entid.film_id_horror
AS
(
    SELECT film_id
    FROM film_category JOIN category USING(category_id)
    WHERE category.name='Horror'
);
```

Fonctions SQL

Dans votre schéma personnel entid (il faut remplacer entid par votre nom de login), 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. La syntaxe est :

```
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
        film_category fc

JOIN
        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')?

② Solution Les solutions sont données en utilisant la fonction suivante CREATE OR REPLACE FUNCTION entid.film_id_cat(s category.name%TYPE) RETURNS TABLE(film_id film.film_id%TYPE) AS \$\$ SELECT fc.film_id FROM film_category fc JOIN category ca ON (fc.category_id=ca.category_id) WHERE ca.name=s; \$\$ LANGUAGE sql;

Solution Solution uniquement avec des jointures : SELECT DISTINCT ac.* FROM actor ac JOIN film_actor fa USING (actor_id) JOIN film_id_cat('Horror') USING (film_id); (156 tuples renvoyés). Solution avec une ETC: with actor_with_horror as select fa.actor_id from film_actor fa where fa.film_id in (select * from film_id_cat('Horror') select distinct ac.* from actor ac join actor_with_horror using (actor_id);

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

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

```
Solution
Réponse avec une jointure externe :
with actor_with_comedy as
    select distinct actor_id
    from film_actor fa
    join film_category fc using (film_id)
    join category c using (category_id)
    where c.name='Comedy'
select distinct last_name, first_name
from actor a left join actor_with_comedy ac using (actor_id)
where ac.actor_id is null
Une autre solution avec EXCEPT
with actor_without_comedy as
(
        select distinct actor_id
        from actor
    )
    except
        select distinct actor_id
        from film_actor
        join film_category using (film_id)
        join category using (category_id)
        WHERE name='Comedy'
    )
)
select last_name, first_name
from actor join actor_without_comedy using (actor_id);
Une autre réponse avec une sous-requête est
SELECT DISTINCT last_name, first_name
FROM
  actor a1
WHERE
 NOT EXISTS
  (SELECT *
  FROM
    film_actor a2
    JOIN film_category using (film_id)
    JOIN category using (category_id)
  WHERE
      a1.actor_id = a2.actor_id
      AND name='Comedy'
  );
```

ou encore (qu'en pensez-vous?):

Solution SELECT DISTINCT ac.last_name, ac_first_name FROM actor ac WHERE NOT EXISTS (SELECT * FROM film_actor fa WHERE film_id IN (SELECT * FROM entid.film_id_cat('Comedy')) AND fa.actor_id = ac.actor_id

- 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')? (129 lignes)
- ♦ Ici l'erreur la plus fréquente consiste à écrire

```
SELECT
  actor_id
FROM
  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 par exemple :

```
\begin{split} \pi_{\texttt{last\_name},\texttt{first\_name}} \bigg( & \texttt{actor} \bowtie \\ & \bigg( \pi_{\texttt{actor\_id}} \big( \texttt{film\_actor} \bowtie \texttt{film\_id\_cat}(\texttt{'Children'}) \big) \bigcap \\ & \pi_{\texttt{actor\_id}} \big( \texttt{film\_actor} \bowtie \texttt{film\_id\_cat}(\texttt{'Horror'}) \big) \bigg) \bigg) \end{split}
```

En SQL, cela peut s'écrire uniquement avec des jointures :

```
SELECT DISTINCT a.first_name, a.last_name
FROM film_id_cat('Horror') fc1
    JOIN film_actor fa1 USING(film_id)
    JOIN actor a USING(actor_id)
    JOIN film_actor fa2 ON a.actor_id = fa2.actor_id
    JOIN film_id_cat('Children') fc2 ON fa2.film_id = fc2.film_id;
```

Ou bien avec des sous-requêtes :