

A14 Subconsultes. Altres sentències SQL.

1) Crear i importar la BBDD

Importa l'arxiu amb la base de dades de prova **sakila.sql**. Aquesta BBDD s'ha d'importar en dues parts: 1er l'esquema i després les dades (insert-data)
Comprova que s'ha importat correctament.

```
valentin@ubuntupc: ~  
postgres@ubuntupc:/home/valentin$ psql  
could not change directory to "/home/valentin": Permiso denegado  
psql (14.11 (Ubuntu 14.11-0ubuntu0.22.04.1))  
Type "help" for help.  
  
postgres=# CREATE DATABASE a14_valentin;  
CREATE DATABASE  
postgres=# exit  
postgres@ubuntupc:/home/valentin$ psql -U postgres -d a14_valentin < /tmp/a14  
could not change directory to "/home/valentin": Permiso denegado  
SET  
SET  
SET  
SET  
SET  
COMMENT  
CREATE EXTENSION
```

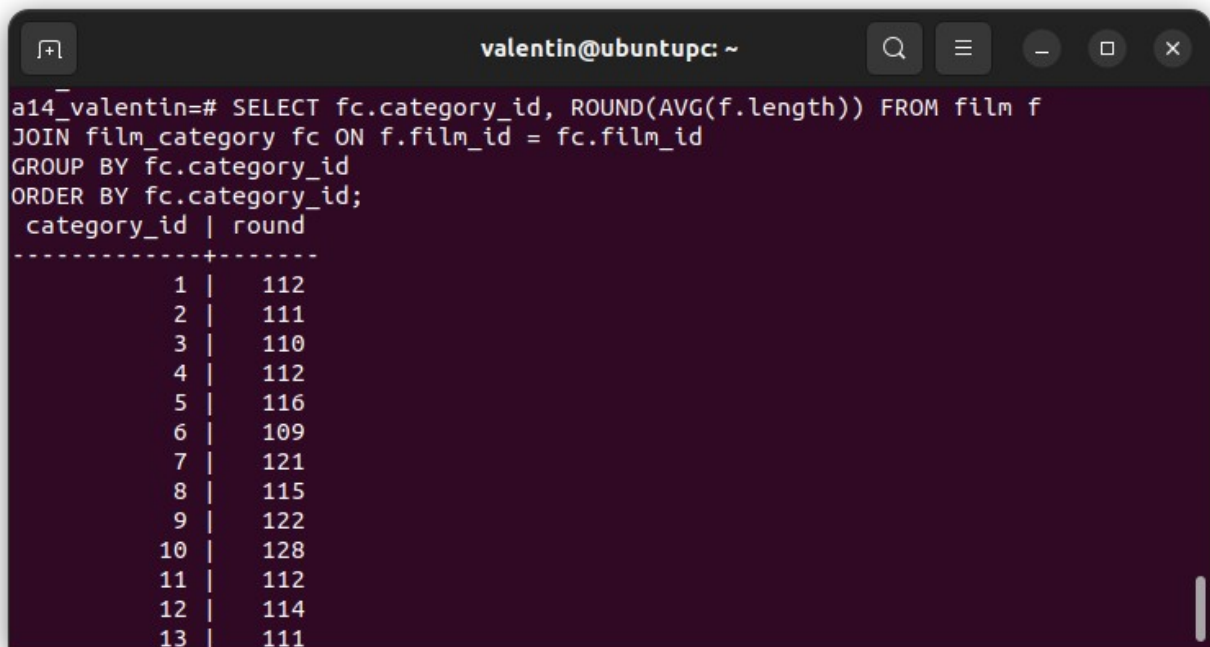
```
valentin@ubuntupc: ~  
postgres@ubuntupc:/home/valentin$ psql -U postgres -d a14_valentin < /tmp/a14_in  
sert  
could not change directory to "/home/valentin": Permiso denegado  
SET  
SET  
SET  
SET  
SET  
SET  
setval  
-----  
200  
(1 row)  
  
setval  
-----  
16  
(1 row)
```

2) Altres subconsultes

Resol les següents consultes amb subconsultes. Recorda que les subconsultes es poden trobar a les següents clàusules/sentències: select, from, where, having i les sentències insert, update o delete.

2.1.) Obténir el promig de duració de pel·lícules per categoria

```
SELECT fc.category_id, ROUND(AVG(f.length)) FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
GROUP BY fc.category_id
ORDER BY fc.category_id;
```



The image shows a terminal window titled 'valentin@ubuntupc: ~'. The user has executed an SQL query to find the average duration of movies by category. The query is: `SELECT fc.category_id, ROUND(AVG(f.length)) FROM film f JOIN film_category fc ON f.film_id = fc.film_id GROUP BY fc.category_id ORDER BY fc.category_id;`. The output is a table with two columns: 'category_id' and 'round'. The results are as follows:

category_id	round
1	112
2	111
3	110
4	112
5	116
6	109
7	121
8	115
9	122
10	128
11	112
12	114
13	111

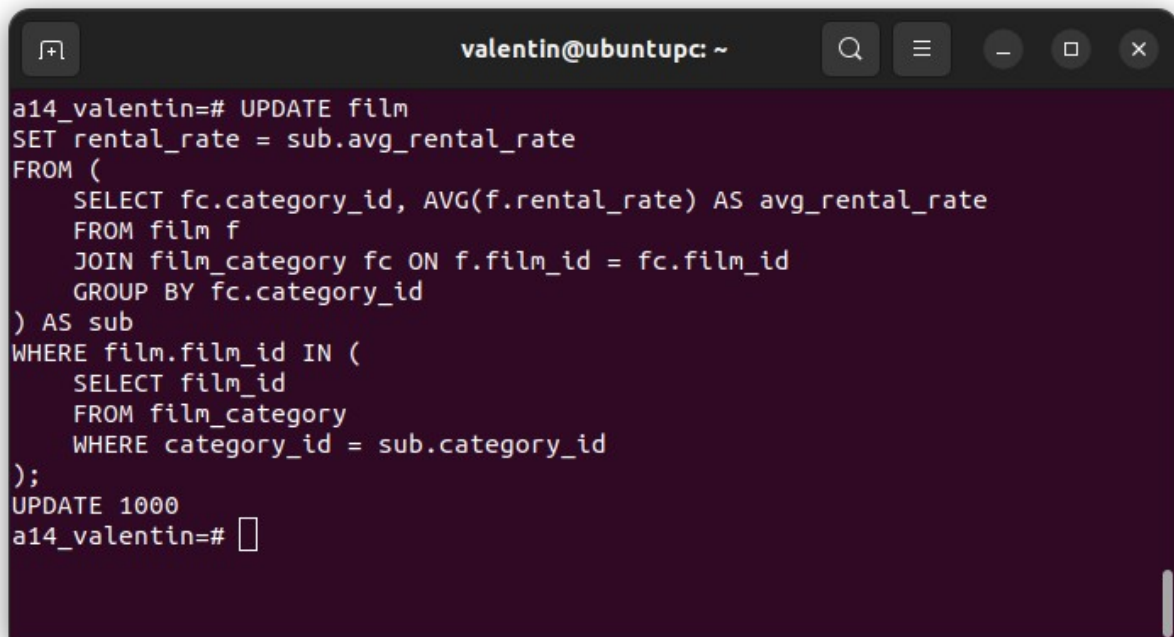
A14 Subconsultes. Altres sentències SQL.

UF2 A14 Altres sentències SQL

2.2.) Trobar les categories on el número total de pel·lícules sigui superior a la mitja de duració per a aquella categoria.

2.3.) Actualitza el preu de lloguer de les pel·lícules per a que sigui igual al promig de preus per a la seva categoria

```
UPDATE film
SET rental_rate = sub.avg_rental_rate
FROM (
    SELECT fc.category_id, AVG(f.rental_rate) AS avg_rental_rate
    FROM film f
    JOIN film_category fc ON f.film_id = fc.film_id
    GROUP BY fc.category_id
) AS sub
WHERE film.film_id IN (
    SELECT film_id
    FROM film_category
    WHERE category_id = sub.category_id
);
```



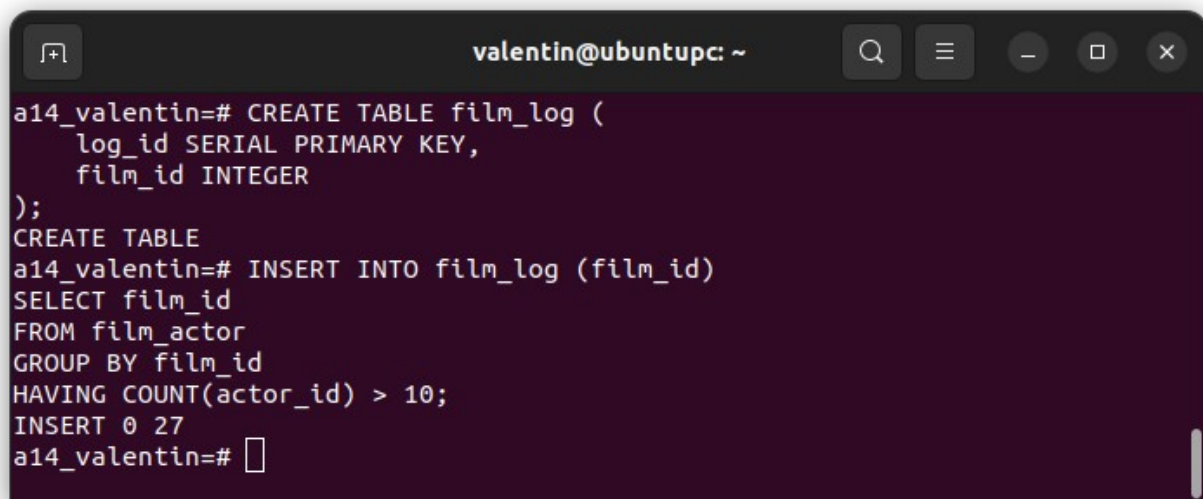
The screenshot shows a terminal window titled 'valentin@ubuntupc: ~'. The terminal displays the following SQL command and its execution:

```
a14_valentin=# UPDATE film
SET rental_rate = sub.avg_rental_rate
FROM (
    SELECT fc.category_id, AVG(f.rental_rate) AS avg_rental_rate
    FROM film f
    JOIN film_category fc ON f.film_id = fc.film_id
    GROUP BY fc.category_id
) AS sub
WHERE film.film_id IN (
    SELECT film_id
    FROM film_category
    WHERE category_id = sub.category_id
);
UPDATE 1000
a14_valentin=#
```

2.4.) Insertar a una tabla de log (per exemple, film_log) les IDs de pel·lícules que tenen més de 10 actors.

```
CREATE TABLE film_log (  
  log_id SERIAL PRIMARY KEY,  
  film_id INTEGER  
);
```

```
INSERT INTO film_log (film_id)  
SELECT film_id  
FROM film_actor  
GROUP BY film_id  
HAVING COUNT(actor_id) > 10;
```



```
valentin@ubuntupc: ~  
a14_valentin=# CREATE TABLE film_log (  
    log_id SERIAL PRIMARY KEY,  
    film_id INTEGER  
);  
CREATE TABLE  
a14_valentin=# INSERT INTO film_log (film_id)  
SELECT film_id  
FROM film_actor  
GROUP BY film_id  
HAVING COUNT(actor_id) > 10;  
INSERT 0 27  
a14_valentin=#
```

2.5.) Eliminar els clients que no han fet lloguer en l'últim any.

La query per eliminar els clients que no han fet lloguer en l'últim any es la següent:

```
DELETE FROM customer  
WHERE customer_id NOT IN (  
    SELECT DISTINCT c.customer_id  
    FROM customer c  
    JOIN payment p ON c.customer_id = p.customer_id  
    WHERE p.payment_date >= CURRENT_DATE - INTERVAL '1 year'  
);
```

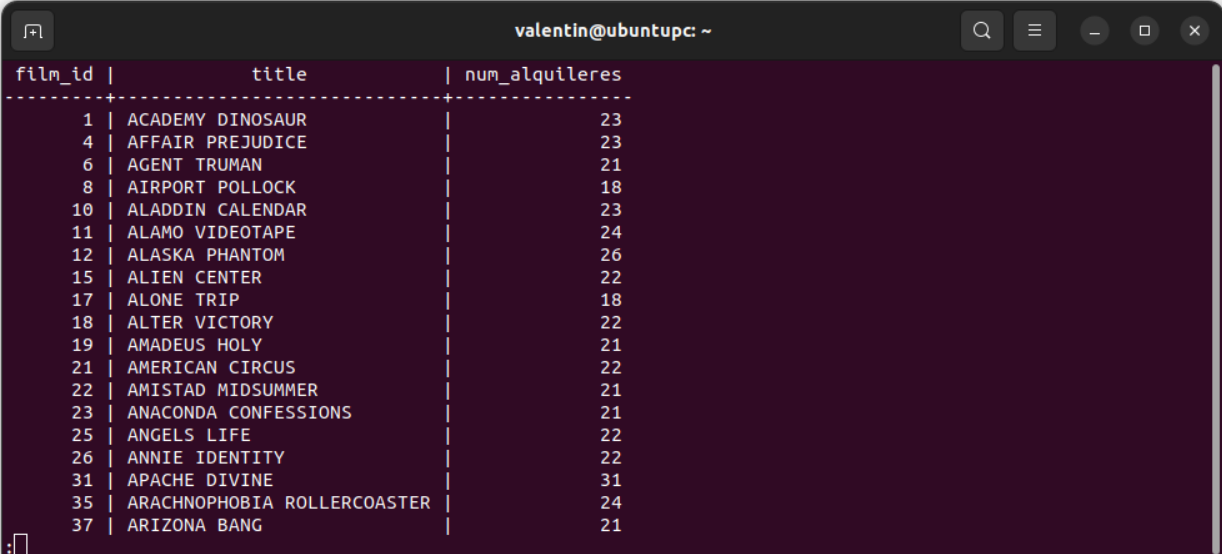
El problema ve que la taula customer està relacionada amb FK's de diverses taules, per exemple la taula payments y la taula rentals, llavors hem de esborrar tots els registres dels clients que no han fet lloguer en l'últim any abans de esborrar la taula customer per a que funcioni.

A14 Subconsultes. Altres sentències SQL.

UF2 A14 Altres sentències SQL

2.6.) Llistar les pel·lícules juntament amb el número de còpies que han estat llogades més que la mitja de lloguers per pel·lícula.

```
SELECT f.film_id, f.title, COUNT(*) AS num_alquileres
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.film_id, f.title
HAVING COUNT(*) > (
SELECT AVG(num_rentals)
FROM (
SELECT COUNT(*) AS num_rentals
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.film_id
) AS subquery
)
ORDER BY f.film_id;
```



A terminal window titled 'valentin@ubuntupc: ~' displays the results of the SQL query. The output is a table with three columns: 'film_id', 'title', and 'num_alquileres'. The table lists 20 movies with their respective rental counts. The window has a dark background and standard Ubuntu window controls.

film_id	title	num_alquileres
1	ACADEMY DINOSAUR	23
4	AFFAIR PREJUDICE	23
6	AGENT TRUMAN	21
8	AIRPORT POLLOCK	18
10	ALADDIN CALENDAR	23
11	ALAMO VIDEOTAPE	24
12	ALASKA PHANTOM	26
15	ALIEN CENTER	22
17	ALONE TRIP	18
18	ALTER VICTORY	22
19	AMADEUS HOLY	21
21	AMERICAN CIRCUS	22
22	AMISTAD MIDSUMMER	21
23	ANACONDA CONFESSIONS	21
25	ANGELS LIFE	22
26	ANNIE IDENTITY	22
31	APACHE DIVINE	31
35	ARACHNOPHOBIA ROLLERCOASTER	24
37	ARIZONA BANG	21

3) Union, intersect, except

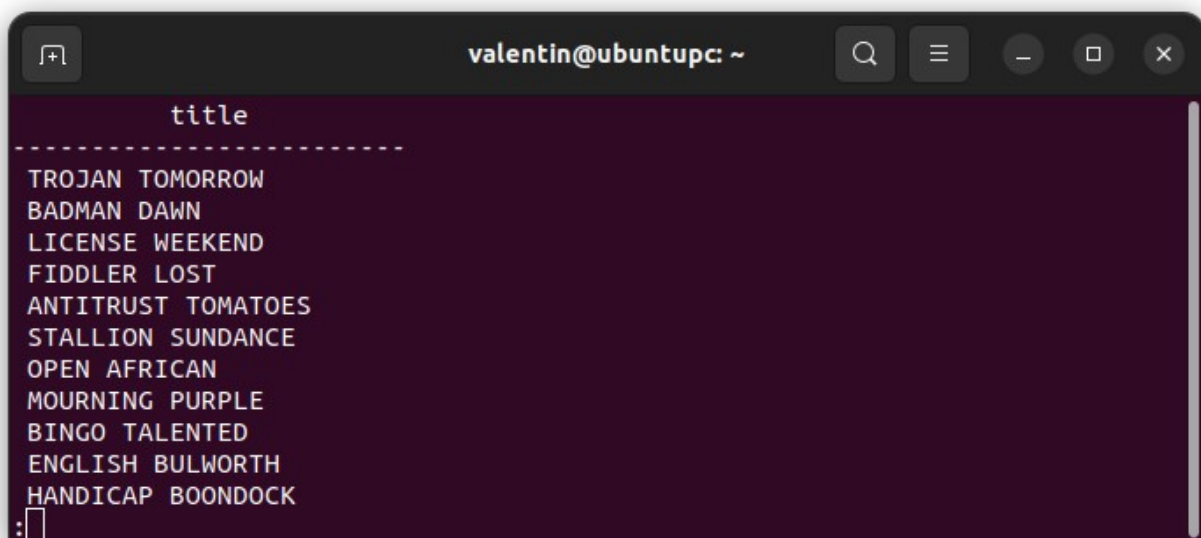
Fes servir les operacions d'unió, intersecció i excepte per a resoldre les següents consultes a la base de dades.

3.1.) Llista totes les pel·lícules de les categories "Acció" i "Ciencia Ficció" sense duplicats.

```
SELECT DISTINCT f.title
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
WHERE c.name = 'Action'
```

UNION

```
SELECT DISTINCT f.title
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
WHERE c.name = 'Sci-Fi';
```



```
valentin@ubuntupc: ~
title
-----
TROJAN TOMORROW
BADMAN DAWN
LICENSE WEEKEND
FIDDLER LOST
ANTITRUST TOMATOES
STALLION SUNDANCE
OPEN AFRICAN
MOURNING PURPLE
BINGO TALENTED
ENGLISH BULWORTH
HANDICAP BOONDOCK
: 
```

A14 Subconsultes. Altres sentències SQL.

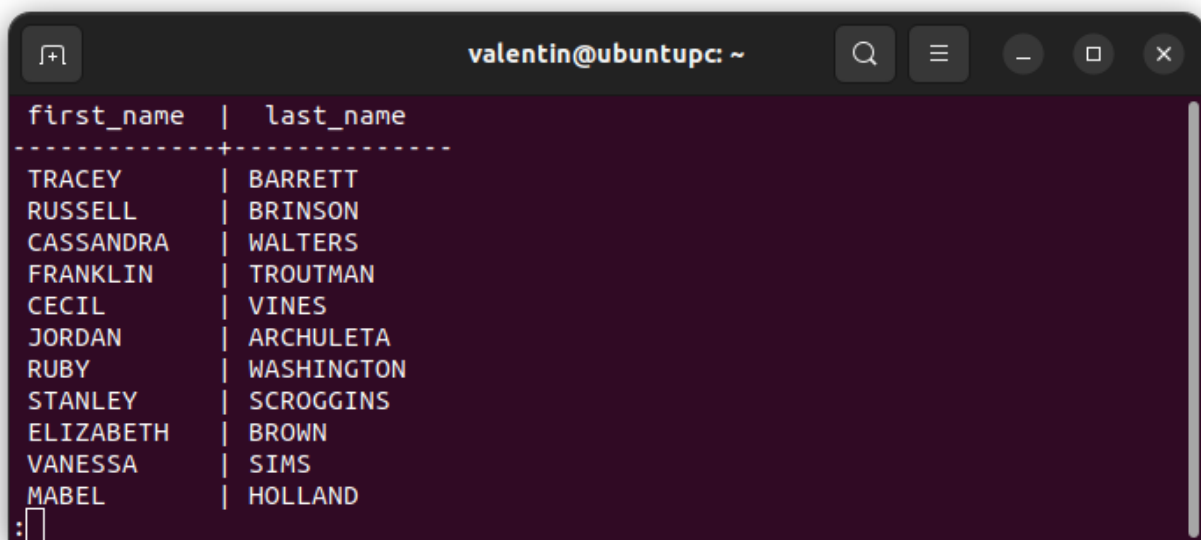
UF2 A14 Altres sentències SQL

3.2.) Troba els noms els clients que han llogat tant pel·lícules d'acció com de romanç.

```
SELECT DISTINCT c.first_name, c.last_name
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category cat ON fc.category_id = cat.category_id
WHERE cat.name = 'Action'
```

INTERSECT

```
SELECT DISTINCT c.first_name, c.last_name
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category cat ON fc.category_id = cat.category_id
WHERE cat.name = 'Drama'; --- No poss-ho Romance perquè no hi existeix
```

A terminal window titled 'valentin@ubuntupc: ~' displays the results of an SQL query. The output is a table with two columns: 'first_name' and 'last_name'. The data is as follows:

first_name	last_name
TRACEY	BARRETT
RUSSELL	BRINSON
CASSANDRA	WALTERS
FRANKLIN	TROUTMAN
CECIL	VINES
JORDAN	ARCHULETA
RUBY	WASHINGTON
STANLEY	SCROGGINS
ELIZABETH	BROWN
VANESSA	SIMS
MABEL	HOLLAND

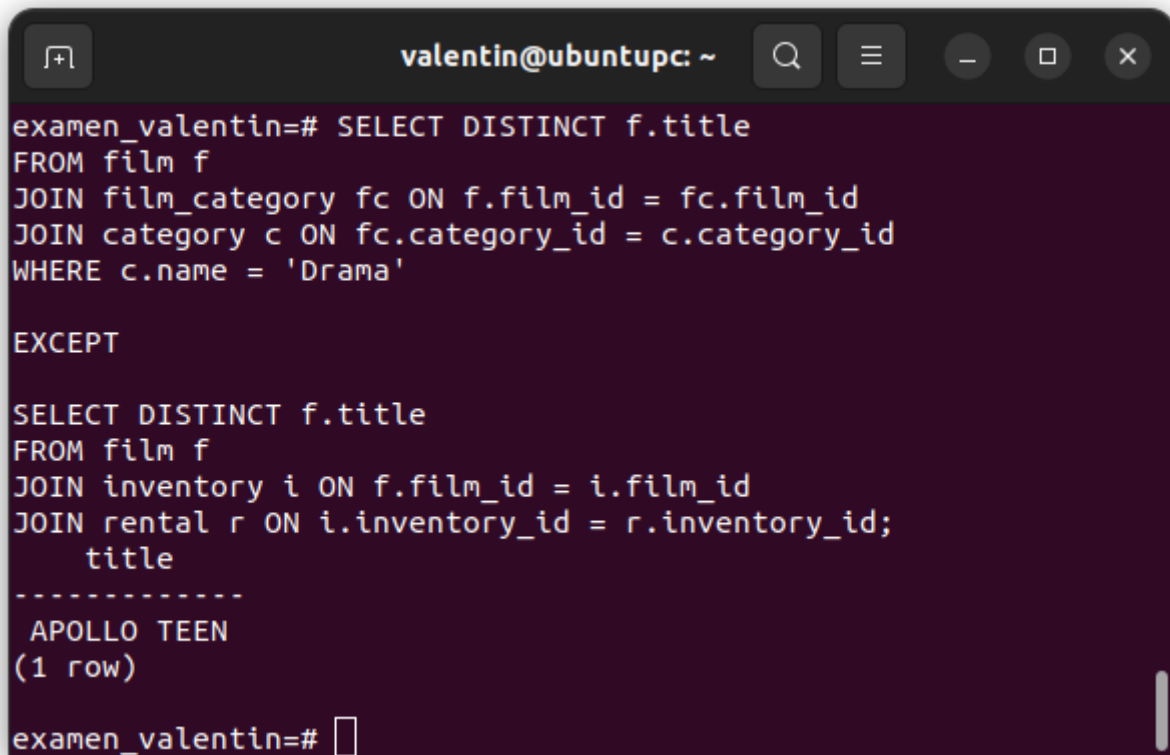
The terminal shows a prompt ':' at the bottom left.

3.3.) Identifica tots els drames (categoria) que no han estat llogats.

```
SELECT DISTINCT f.title
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
WHERE c.name = 'Drama'
```

EXCEPT

```
SELECT DISTINCT f.title
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id;
```

A terminal window titled 'valentin@ubuntupc: ~' with standard Ubuntu window controls. It displays two SQL queries. The first query, 'SELECT DISTINCT f.title FROM film f JOIN film_category fc ON f.film_id = fc.film_id JOIN category c ON fc.category_id = c.category_id WHERE c.name = 'Drama'', is followed by the word 'EXCEPT'. The second query, 'SELECT DISTINCT f.title FROM film f JOIN inventory i ON f.film_id = i.film_id JOIN rental r ON i.inventory_id = r.inventory_id;', is followed by the result 'APOLLO TEEN' and '(1 row)'. The prompt 'examen_valentin=#' is visible at the bottom.

```
examen_valentin=# SELECT DISTINCT f.title
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
WHERE c.name = 'Drama'

EXCEPT

SELECT DISTINCT f.title
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id;
      title
-----
  APOLLO TEEN
(1 row)

examen_valentin=#
```


4) MERGE

L'empresa de lloguer de pel·lícules en que es basa la bbdd sakila té locals d'una altra empresa recentment adquirida que envia la informació dels seus lloguers a la bbdd central on s'ha de consolidar.

Per aquest motiu existeix la taula external_rentals amb les dades dels lloguers d'altres tendes. La taula external_rentals té les següents columnes:

ext_rental_id (suposem que aquest ID és únic i proporcionat per l'empresa externa), rental_date i return_date (TIMESTAMP), inventory_id, customer_id, staff_id (enter).

4.1.) Crea una sentència MERGE que insereixi nous registres a la taula de lloguers (rental) si no existeixen. En cas que existeixi el registre, s'ha d'actualitzar la data de retorn.

```
MERGE INTO rental AS target
USING external_rentals AS source
ON target.ext_rental_id = source.ext_rental_id
WHEN MATCHED THEN
UPDATE SET target.return_date = source.return_date
WHEN NOT MATCHED THEN
INSERT (rental_id, rental_date, inventory_id, customer_id, return_date, staff_id)
VALUES (source.ext_rental_id, source.rental_date, source.inventory_id, source.customer_id,
source.return_date, source.staff_id);
```

4.2.) Inserta registres de prova i comproba que la sentència MERGE funciona correctament. Inclou les captures corresponents.

```
INSERT INTO external_rentals (ext_rental_id, rental_date, return_date, inventory_id,
customer_id, staff_id)
VALUES
(1, '2024-03-20 10:00:00', '2024-03-22 12:00:00', 1, 1, 1),
(2, '2024-03-21 11:00:00', '2024-03-23 13:00:00', 2, 2, 2),
(3, '2024-03-22 12:00:00', '2024-03-24 14:00:00', 3, 3, 3);
```

Ponemos de nuevo el codigo del MERGE.

5) Consultes externes (LEFT, RIGHT, OUTER)

5.1.) Troba totes les pel·lícules i mostra la seva quantitat de lloguers. Inclou aquelles pel·lícules que mai han estat llogades.

```
SELECT f.title, COALESCE(COUNT(r.rental_id), 0) AS cantidad_alquileres
FROM films f
LEFT JOIN rentals r ON f.film_id = r.film_id
GROUP BY f.title;
```

5.2.) Llista tots els clients i la última data de lloguer. Per aquells clients que no han fet cap lloguer, ha d'aparèixer NULL.

```
SELECT c.customer_id, c.first_name, c.last_name, MAX(r.rental_date) AS
ultima_fecha_alquiler
FROM customers c
LEFT JOIN rentals r ON c.customer_id = r.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name;
```

5.3.) Mostra totes les pel·lícules i clients, incloent aquells que no han llogat pel·lícules i aquelles pel·lícules que no han estat llogades.

```
SELECT f.title AS pelicula, c.first_name || ' ' || c.last_name AS cliente
FROM films f
CROSS JOIN customers c
LEFT JOIN rentals r ON f.film_id = r.film_id AND c.customer_id = r.customer_id;
```

6) Clàusules avançades: funcions WINDOW, CTE

6.1.) Indica que és una funció de ventana i la seva relació amb les funcions de resum amb les teves paraules (sense copiar/pegar). Posa un exemple vàlid per a la bbdd sakila.

Una función de ventana es básicamente una herramienta que te permite realizar cálculos sobre un conjunto específico de filas en tus datos. En este caso, podrías usar una función de ventana para ordenar todas las películas por su duración y asignarles un rango en función de esa clasificación.

Por ejemplo, si tienes una lista de películas y quieres asignarles un rango basado en cuán largas son, una función de ventana te ayudaría a hacer exactamente eso. Te permitiría clasificar cada película según su duración y asignarles un rango en función de cuán largas son en comparación con las demás.

```
SELECT
title,
length,
RANK() OVER (ORDER BY length DESC) AS ranking_por_duracion
FROM
film;
```

6.2.) Indica que és un CTE i posa un exemple per a la bbdd fent servir la clàusula WITH.

Es una expresión temporal que se define dentro de una consulta SQL y se utiliza para crear un conjunto de datos temporal que puede ser referenciado posteriormente dentro de esa misma consulta.

```
WITH DuracionPromedioPorCategoria AS (
SELECT
c.name AS categoria,
AVG(f.length) AS duracion_promedio
FROM
film f
JOIN
film_category fc ON f.film_id = fc.film_id
JOIN
category c ON fc.category_id = c.category_id
GROUP BY
c.name
)
SELECT
categoria,
duracion_promedio
FROM
DuracionPromedioPorCategoria;
```

7) XML

7.1) Guarda la taula actors en un arxiu xml. Fes servir les funcions xmlelement(), xmlagg() i xmlforest().

```
SELECT xmlelement(name "actors",
xmlagg(
xmlelement(name "actor",
xmlforest(
actor_id AS "actor_id",
first_name AS "first_name",
last_name AS "last_name"
)
)
) AS xml_data
FROM actors;
```

7.2) Importa les dades dels actors des d'un arxiu xml fins a una taula temporal d'importació. La taula a importar (new_actor) ha de tenir els camps actor_id (enter), first_name i last_name (cadena variable). El format de l'xml és el següent:

```
<actors>
  <actor>
    <id>101</id>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </actor>
  <actor>
    ...
  </actor>
</actors>
```

```
CREATE TEMP TABLE new_actor (
actor_id INT,
first_name VARCHAR,
last_name VARCHAR
);
```

```
INSERT INTO new_actor (actor_id, first_name, last_name)
SELECT
(xpath('/actor/id/text()', xml_data))[1]::text::int AS actor_id,
```

A14 Subconsultes. Altres sentències SQL.**UF2 A14 Altres sentències SQL**

```
(xpath('/actor/firstName/text()', xml_data))[1]::text AS first_name,  
(xpath('/actor/lastName/text()', xml_data))[1]::text AS last_name  
FROM  
XMLTABLE('/actors/actor' PASSING  
XMLPARSE(DOCUMENT convert_from(pg_read_binary_file('actores.xml'), 'UTF8')) AS  
xml_data  
);
```

Crea varis registres de prova i deixa l'arxiu xml actors.xml al directori de treball de Postgres (comanda show data_directory).

Una vegada tinguis l'arxiu amb alguns registres d'exemple i la taula creada, executa el següent **bloc anònim** de codi PL/pgSQL.

```
DO $$  
DECLARE  
    data xml;  
    actor_record record;  
BEGIN  
    -- Carregar el contingut de l'arxiu XML a 'data'  
    data := pg_read_file('actors.xml', 0, 100000); -- Ajusta la  
ruta  
  
    -- Iterar a través dels elements de l'XML  
    FOR actor_record IN  
        SELECT  
            (xpath('//id/text()', actor))[1]::text::int AS  
actor_id,  
            (xpath('//firstName/text()', actor))[1]::text AS  
first_name,  
            (xpath('//lastName/text()', actor))[1]::text AS  
last_name  
        FROM  
            unnest(xpath('//actor', data)) AS t(actor)  
    LOOP  
        INSERT INTO new_actor (actor_id, first_name, last_name)  
        VALUES (actor_record.actor_id, actor_record.first_name,  
actor_record.last_name);  
    END LOOP;  
END $$;
```

Comproba que s'han importat les dades correctament i adjunta les captures que consideris.

8) Consulta el diccionari de dades.

Obteniu les següents dades consultant el diccionari de dades enlloc de les comandes \.

8.1.) Llistar totes les bases de dades disponibles a la teva instància de Postgres

```
SELECT datname
FROM pg_database
WHERE datistemplate = false;
```

8.2.) Mostrar totes les taules relacionades amb un esquema específic (per exemple públic). Mostrar també totes les vistes del mateix esquema. Pots fer dues consultes o una consulta combinant els resultats de les taules i les vistes.

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
AND table_type = 'BASE TABLE';
```

UNION

```
SELECT table_name
FROM information_schema.views
WHERE table_schema = 'public';
```

8.3.) Obtenir les columnes d'una taula determinada

```
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'film';
```

8.4.) Mostra les restriccions de clau primària a una base de dades.

```
SELECT constraint_name, table_name, column_name
FROM information_schema.key_column_usage
WHERE constraint_name LIKE '%_pkey';
```

Rúbrica avaluació

Criteri avaluació	0	1	2
	No fet	Crea vistes bàsiques i sinònims amb alguns errors importants.	Crea vistes complexes correctament
	No fet	No fa servir l'eina gràfica Dbeaver o té omisions importants.	Crea vistes amb Dbeaver correctament
	No fet	Crea vistes bàsiques amb alguns errors importants.	Crea vistes complexes correctament
	No fet	Crea vistes bàsiques amb joins que no funcionen correctament.	Crea vistes bàsiques amb joins i aquestes són correctes
	No fet	Crea vistes materialitzades amb errors o no s'actualitzen.	Crea vistes materialitzades i aquestes s'actualitzen correctament