Nombre de la práctica	Operadores set, joins y subconsultas			No.	01
Asignatura:	Taller de bases de datos	Carrera:	Ingeniería en sistemas	Duración de la práctica (Hrs)	

Nombre del alumno: Jesús Navarrete Martínez
Grupo: 3501

I. Competencia(s) específica(s):

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro): Aula

III. Material empleado:

- **Equipo de computo**

IV. Desarrollo de la práctica:

Consulta 1:

SELECT AVG(LENGTH(first_name)) FROM actor;

- **Tipo:** Subconsulta independiente.
- **Descripción:** Calcula el promedio de la longitud de los nombres (first_name) de los actores en la tabla actor. Esta consulta devuelve un solo valor: el promedio de caracteres de todos los first_name de los actores.

Consulta 2:

SELECT * FROM actor WHERE LENGTH(first_name) > 5.3050;

- **Tipo:** Consulta simple.
- **Descripción:** Selecciona todos los registros de la tabla actor donde la longitud del nombre (first_name) es mayor a 5.3050 caracteres. La precisión decimal en 5.3050 es irrelevante para este contexto, ya que la longitud es un número entero, pero MySQL lo permite. La consulta devuelve todos los actores con nombres más largos que 5 caracteres.

Consulta 3:

SELECT * FROM actor WHERE LENGTH(first_name) > (SELECT AVG(LENGTH(first_name)) FROM actor);

- **Tipo:** Subconsulta correlacionada.
- **Descripción:** Selecciona todos los registros de la tabla actor donde la longitud del nombre (first_name) es mayor que el promedio de longitud de los nombres de los actores. La subconsulta (SELECT AVG(**LENGTH**(first_name)) FROM actor) calcula el promedio de longitud de first_name, y luego el SELECT principal utiliza este valor como referencia para filtrar a los actores con nombres más largos que el promedio.

Consulta 4:

SELECT category_id FROM category WHERE name="Comedy";

- **Tipo:** Consulta simple.
- **Descripción:** Busca el category_id de la categoría de películas con el nombre "Comedy" en la tabla category. Retorna el identificador único de la categoría "Comedy".

Consulta 5:

SELECT film_id FROM film_category WHERE category_id=5;

- **Tipo:** Consulta simple.
- **Descripción:** Obtiene los film_id de las películas que pertenecen a la categoría cuyo category_id es 5 en la tabla film_category. En este caso, se asume que la categoría "Comedy" tiene un category_id de 5. Devuelve los IDs de las películas en esta categoría.

Consulta 6:

SELECT film_id FROM film_category WHERE category_id=(SELECT category_id FROM category WHERE name="Comedy");

- **Tipo:** Subconsulta.
- **Descripción**: Encuentra los film_id de las películas que pertenecen a la categoría "Comedy" sin suponer un valor específico de category_id. La subconsulta (SELECT category_id FROM category WHERE name="Comedy") busca el category_id de "Comedy" y lo usa para filtrar las películas en film_category.

Consulta 7:

SELECT actor_id FROM film_actor WHERE film_id IN (7,28,99);

- **Tipo:** Consulta con condición IN.
- **Descripción:** Obtiene los actor_id de los actores que participaron en las películas con los film_id específicos 7, 28 y 99. Esta consulta es útil para ver qué actores participaron en películas específicas.

Consulta 8:

SELECT actor_id FROM film_actor WHERE film_id IN (SELECT film_id FROM film_category WHERE category_id=(SELECT category_id FROM category WHERE name="Comedy"));

- **Tipo:** Subconsulta anidada.
- **Descripción:** Encuentra los actor_id de los actores que participaron en películas de la categoría "Comedy". Primero se obtiene el category_id de "Comedy", luego se encuentran los film_id correspondientes, y finalmente se obtienen los actor_id de los actores que actuaron en esas películas.

Consulta 9:

SELECT first_name, last_name FROM actor WHERE actor_id IN (SELECT actor_id FROM film_actor WHERE film_id IN (SELECT film_id FROM film_category WHERE category_id=(SELECT category_id FROM category WHERE name="Comedy")));

- **Tipo:** Subconsulta anidada.
- **Descripción:** Selecciona los nombres (first_name, last_name) de los actores que han participado en películas de la categoría "Comedy". Utiliza subconsultas para, en primer lugar, obtener el category_id de "Comedy", luego los film_id de las películas en esa categoría, y finalmente los actor_id de los actores que trabajaron en dichas películas.

Consulta 10:

SELECT inventory_id FROM rental;

- **Tipo:** Consulta simple.
- **Descripción:** Selecciona todos los inventory_id de la tabla rental, que representa los artículos (copias de películas) que han sido alquilados al menos una vez.

Consulta 11:

SELECT film_id FROM inventory WHERE inventory_id IN (SELECT inventory_id FROM rental);

- **Tipo:** Subconsulta.
- **Descripción:** Encuentra los film_id de las películas que han sido alquiladas al menos una vez. Utiliza la subconsulta (SELECT inventory_id FROM rental) para obtener los inventory_id que han sido rentados, y luego recupera los film_id correspondientes en la tabla inventory.

Consulta 12:

SELECT title FROM film WHERE film_id NOT IN (SELECT film_id FROM inventory WHERE inventory_id IN (SELECT inventory_id FROM rental));

- **Tipo:** Subconsulta anidada.
- **Descripción:** Selecciona los títulos (title) de las películas que nunca han sido alquiladas. Utiliza subconsultas anidadas para identificar primero los inventory_id que han sido alquilados y luego los film_id correspondientes. Finalmente, el NOT IN asegura que solo se seleccionen aquellos film_id que no aparecen en la lista de películas rentadas, es decir, aquellas que nunca fueron alquiladas.

Consulta 13:

```
SELECT f.title, c.name AS category_name
 FROM film AS f
 INNER JOIN film_category AS fc
 ON f.film_id = fc.film_id
 INNER JOIN category AS c
 ON fc.category_id = c.category_id;
```

- **Tipo:** JOIN (Específicamente, INNER JOIN).
- **Descripción:** Esta consulta obtiene los títulos de las películas (f.title) junto con el nombre de su categoría (c.name).
  - Primero, se realiza un INNER JOIN entre la tabla film (f) y la tabla film_category (fc) usando la columna film_id. Esto permite vincular cada película con su categoría correspondiente.
  - Luego, se hace un segundo INNER JOIN entre film_category y category (c) a través de category_id, para obtener el nombre de la categoría.

  El resultado muestra todas las películas junto con el nombre de la categoría a la que pertenecen. Solo se muestran películas que tienen al menos una categoría, ya que el INNER JOIN excluye las películas sin una categoría asociada.

Consulta 14:

```
SELECT * FROM film
INNER JOIN film_category ON film.film_id = film_category.film_id
JOIN category AS c ON film_category.category_id = c.category_id;
```

- **Tipo:** JOIN (Específicamente, INNER JOIN).
- **Descripción:** Selecciona todos los campos (*) de las tablas film, film_category, y category para todas las películas que están asociadas con una categoría.
  - Primero, se realiza un INNER JOIN entre film y film_category utilizando film_id, lo que permite enlazar cada película con su categoría asignada.
  - Luego, se usa otro JOIN (que es un INNER JOIN por defecto) para enlazar film_category con la tabla category mediante category_id, obteniendo así los detalles de la categoría.

- El resultado incluye todos los datos de las películas que tienen categorías, así como los detalles de estas categorías.

Consulta 15:

```
SELECT film.title, c.name FROM film
INNER JOIN film_category ON film.film_id = film_category.film_id
JOIN category AS c ON film_category.category_id = c.category_id;
```

- **Tipo:** JOIN (Específicamente, INNER JOIN).
- **Descripción:** Selecciona únicamente los títulos de las películas (film.title) y los nombres de sus categorías (c.name).
  - Al igual que en la primera consulta, enlaza las tablas film, film_category, y category mediante INNER JOIN para obtener solo las películas con categorías.
- Esta consulta es más específica, ya que solo devuelve los nombres de las películas y de sus categorías, en lugar de todos los datos de las tablas.

Consulta 16:

```
SELECT f.title, COUNT(r.rental_id) AS cantidad_rentas
 FROM film AS f
 LEFT JOIN inventory i ON f.film_id = i.film_id
 LEFT JOIN rental AS r ON i.inventory_id = r.inventory_id
 GROUP BY f.title;
```

- **Tipo:** JOIN (Específicamente, LEFT JOIN), con GROUP BY.
- **Descripción:** Esta consulta obtiene el título de cada película (f.title) junto con la cantidad de veces que ha sido alquilada (cantidad_rentas).
  - Se realiza un primer LEFT JOIN entre film (f) y inventory (i) utilizando la columna film_id, para obtener todos los registros de la tabla film y los registros coincidentes en inventory. Esto asegura que todas las películas sean incluidas, incluso si no tienen inventarios asociados.
  - Luego, se hace un segundo LEFT JOIN entre inventory y rental (r) utilizando inventory_id, para incluir todas las películas con sus respectivas rentas. Esto permite que aparezcan las películas aunque no hayan sido alquiladas, ya que el LEFT JOIN incluirá valores NULL en estos casos.
  - Finalmente, GROUP BY f.title agrupa el resultado por título de película, y COUNT(r.rental_id) cuenta la cantidad de veces que la película ha sido rentada. Si una película nunca fue alquilada, la cantidad de rentas será 0.

- **Resultado:** La consulta muestra el título de cada película junto con el número de veces que ha sido alquilada, incluyendo las películas que no tienen rentas (en cuyo caso, cantidad_rentas será 0).

Consulta 17:

```
SELECT a.first_name, a.last_name, f.title
FROM actor AS a
JOIN film_actor AS fa ON a.actor_id = fa.actor_id
JOIN film AS f ON fa.film_id = f.film_id
ORDER BY a.first_name, a.last_name;
```

- **Tipo:** JOIN (Específicamente, INNER JOIN).
- **Descripción:** Esta consulta muestra los nombres (first_name, last_name) de los actores y los títulos (title) de las películas en las que han actuado.
  - Primero, se une la tabla actor (a) con film_actor (fa) usando actor_id, vinculando cada actor con los registros de las películas en las que ha participado.
  - Luego, se realiza otro INNER JOIN entre film_actor y film (f) utilizando film_id para obtener los títulos de las películas.

- **Resultado:** Muestra únicamente los actores que han participado en al menos una película, junto con los títulos de esas películas. La lista está ordenada por el nombre y apellido del actor.

Consulta 18:

```
SELECT a.first_name, a.last_name, f.title
FROM actor AS a
LEFT JOIN film_actor AS fa ON a.actor_id = fa.actor_id
LEFT JOIN film AS f ON fa.film_id = f.film_id;
```

- **Tipo:** LEFT JOIN.
- **Descripción:** Esta consulta muestra todos los actores (first_name, last_name) y los títulos de las películas en las que han participado. Si un actor no ha participado en ninguna película, aún se mostrará su nombre, pero el campo del título (f.title) aparecerá como NULL.
  - Primero, se hace un LEFT JOIN entre actor y film_actor para incluir todos los actores, aunque no tengan películas asociadas.
  - Luego, se realiza otro LEFT JOIN con film para obtener los títulos de las películas, incluyendo los actores que no han actuado en ninguna película.

- **Resultado:** Muestra todos los actores, y en caso de que un actor no haya actuado en ninguna película, el campo title será NULL.

Consulta 19:

```
SELECT a.first_name, a.last_name, f.title
FROM actor AS a
RIGHT JOIN film_actor AS fa ON a.actor_id = fa.actor_id
RIGHT JOIN film AS f ON fa.film_id = f.film_id;
```

- **Tipo:** RIGHT JOIN.
- **Descripción:** Esta consulta muestra los actores y las películas, incluyendo todas las películas incluso si no tienen un actor asociado.
  - El primer RIGHT JOIN entre actor y film_actor asegura que todos los registros de film_actor (cada relación actor-película) se incluyan, aunque no haya coincidencias en actor.
  - El segundo RIGHT JOIN con film garantiza que se incluyan todas las películas, incluso si no tienen actores registrados.

- **Resultado:** Muestra todas las combinaciones de actores y películas, e incluye películas sin actores asociados (donde first_name y last_name serán NULL si no hay actor registrado).

Consulta 20:

```
SELECT first_name FROM actor
UNION
SELECT first_name FROM customer;
```

- **Tipo:** Unión (UNION).
- **Descripción:** Esta consulta combina los nombres (first_name) de los actores y los clientes, eliminando duplicados.
  - El operador UNION toma el conjunto de nombres de la columna first_name en la tabla actor y el conjunto de nombres de la misma columna en la tabla customer.
  - Al usar UNION, se eliminan automáticamente los nombres duplicados en el resultado, mostrando solo valores únicos de first_name en ambas tablas.

- **Resultado:** Devuelve una lista única de nombres (first_name) presentes en ambas tablas (actor y customer), sin repeticiones.

Consulta 21:

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

- **Tipo:** Exclusión (EXCEPT).
- **Descripción:** Esta consulta busca las películas que no han sido alquiladas.
  - La primera parte, SELECT title FROM film, selecciona todos los títulos de las películas en la tabla film.

La segunda parte,

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

utiliza JOIN para obtener los títulos de las películas que sí han sido alquiladas al menos una vez. Este subgrupo incluye únicamente las películas que aparecen en las tablas inventory y rental.
El operador EXCEPT devuelve los títulos de las películas que están en el primer conjunto (todas las películas) pero no en el segundo (películas alquiladas).

- **Resultado:** Muestra una lista de los títulos de las películas que nunca han sido alquiladas.

Consulta 22:

```
SELECT DISTINCT c.city
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id = a.address_id
UNION
SELECT DISTINCT c.city
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN staff s ON s.address_id = a.address_id;
```

- **Tipo:** Unión (UNION) con JOIN.
- **Descripción:** Esta consulta devuelve una lista única de las ciudades en las que viven los clientes o empleados. Utiliza el operador UNION para combinar los resultados de dos consultas, eliminando duplicados entre ambas.

La primera parte de la consulta:

```
SELECT DISTINCT c.city
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id = a.address_id
```

selecciona las ciudades en las que viven los clientes.

- JOIN address a ON c.city_id = a.city_id conecta la tabla city con address para obtener la dirección de cada ciudad.
- JOIN customer cu ON cu.address_id = a.address_id une la dirección con el cliente que reside allí.
- DISTINCT asegura que no haya duplicados dentro de esta parte de la consulta.

La segunda parte de la consulta:

```
SELECT DISTINCT c.city
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN staff s ON s.address_id = a.address_id;
```

selecciona las ciudades en las que viven los empleados, de manera similar, mediante JOIN para vincular las tablas city, address y staff.
Finalmente, UNION combina los resultados de ambas consultas y elimina cualquier ciudad duplicada entre los clientes y los empleados.

- **Resultado:** Una lista única de ciudades donde residen los clientes o empleados, sin duplicados.

Consulta 22:

```
SELECT DISTINCT c.city
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id = a.address_id
UNION
SELECT city FROM city WHERE city_id IN (
   SELECT city_id FROM address WHERE address_id IN (
      SELECT address_id FROM customer
   )
);
```

- **Tipo:** Unión (UNION) y Subconsultas.
- **Descripción:** Esta consulta devuelve una lista única de las ciudades donde viven los clientes. Utiliza el operador UNION para combinar los resultados de dos enfoques diferentes: un JOIN en la primera parte y una serie de subconsultas anidadas en la segunda parte.

Primera Parte:

```
SELECT DISTINCT c.city
FROM city c
JOIN address a ON c.city_id = a.city_id
JOIN customer cu ON cu.address_id = a.address_id
```

- **Tipo:** JOIN.
- **Descripción:** Selecciona ciudades únicas (DISTINCT c.city) donde residen los clientes, utilizando JOIN para vincular las tablas city, address y customer.

Segunda Parte:

```
SELECT city FROM city WHERE city_id IN (
   SELECT city_id FROM address WHERE address_id IN (
      SELECT address_id FROM customer
   )
);
```

- **Tipo:** Subconsultas anidadas.
- **Descripción:** Esta sección utiliza subconsultas para obtener las ciudades donde viven los clientes, de la siguiente forma:
  - La subconsulta más interna (SELECT address_id FROM customer) recupera los address_id de todos los clientes.
  - La subconsulta en el medio (SELECT city_id FROM address WHERE address_id IN (...)) usa esos address_id para obtener los city_id asociados.
  - La consulta externa (SELECT city FROM city WHERE city_id IN (...)) usa los city_id obtenidos para seleccionar el nombre de la ciudad correspondiente.
- Esta parte consigue el mismo resultado que la primera parte, pero utilizando solo subconsultas en lugar de JOIN.
- **Resultado:** Muestra una lista única de ciudades donde residen los clientes, sin duplicados.

V. Conclusiones:

Durante este conjunto de prácticas, exploramos y aplicamos diversos conceptos de SQL, tales como JOIN, UNION, EXCEPT, subconsultas y la creación de vistas (VIEW), para obtener información específica y organizada en la base de datos Sakila. Estos conceptos son esenciales para manejar consultas complejas y optimizar la recuperación de datos de manera estructurada y eficiente.

1. **JOIN y UNION**: Aprendimos cómo usar los JOIN (especialmente INNER, LEFT y RIGHT JOIN) para combinar datos de múltiples tablas basados en condiciones específicas, y cómo UNION nos permite unir resultados de diferentes consultas eliminando duplicados. Estas técnicas permiten agrupar datos de diversas fuentes, dando una vista consolidada de la información.
2. **Subconsultas y EXCEPT**: Con las subconsultas anidadas y el operador EXCEPT, identificamos registros únicos y datos faltantes en conjuntos. Las subconsultas ofrecen una forma de seleccionar datos en base a resultados parciales de otras consultas, mientras que EXCEPT facilita encontrar elementos en una tabla que no están presentes en otra, ideal para análisis comparativo.
3. **Vistas (VIEW)**: La creación de vistas nos ayuda a simplificar y reutilizar consultas complejas, presentando los datos de forma dinámica y accesible para consultas posteriores. Las vistas actúan como tablas virtuales que reflejan cambios en tiempo real según se actualizan los datos subyacentes, brindando una herramienta poderosa para manejar reportes y vistas específicas sin duplicar información.
4. **Inserciones Condicionales**: Al realizar inserciones en las tablas base y observar cómo impactan las vistas, comprendimos cómo los registros cumplen (o no) con las condiciones de una vista y cómo estas se actualizan automáticamente. Esto permite que los datos nuevos sean visibles de inmediato cuando cumplen con criterios específicos.