Consultas a la Base de Datos

Consultas previas para comprobar que incluye Sakila

Para iniciar la ronda de consultas, nos ubicamos en un lienzo nuevo

USE Sakila;

Para que el sistema entienda sobre que base de datos vamos a realizar las consultas.

De esta manera trabajamos sobre la base de datos Sakila de manera directa.

Luego podemos **Verificar las tablas**

SHOW TABLES;

Realizar consultas, por ejemplo, ver los primeros 5 actores

```
SELECT * FROM actor LIMIT 5;
```

A. Nivel inicial:

1. **Listar todas las películas**:
   Nos muestra título y año de lanzamiento de todas las películas en la base de datos.

   -- 1. Listar todas las películas

   -- Esta consulta selecciona todas las películas y sus años de lanzamiento.

   -- a) Selecciona los campos 'title' y 'release_year' de la tabla 'film'.

   SELECT title, release_year FROM film;

2. **Obtener clientes activos:**
   Nos generará una lista con nombre y apellidos de los clientes que tienen en estado la indicación de activos.

   -- 2. Obtener clientes activos

   -- Esta consulta obtiene los nombres y apellidos de los clientes activos.

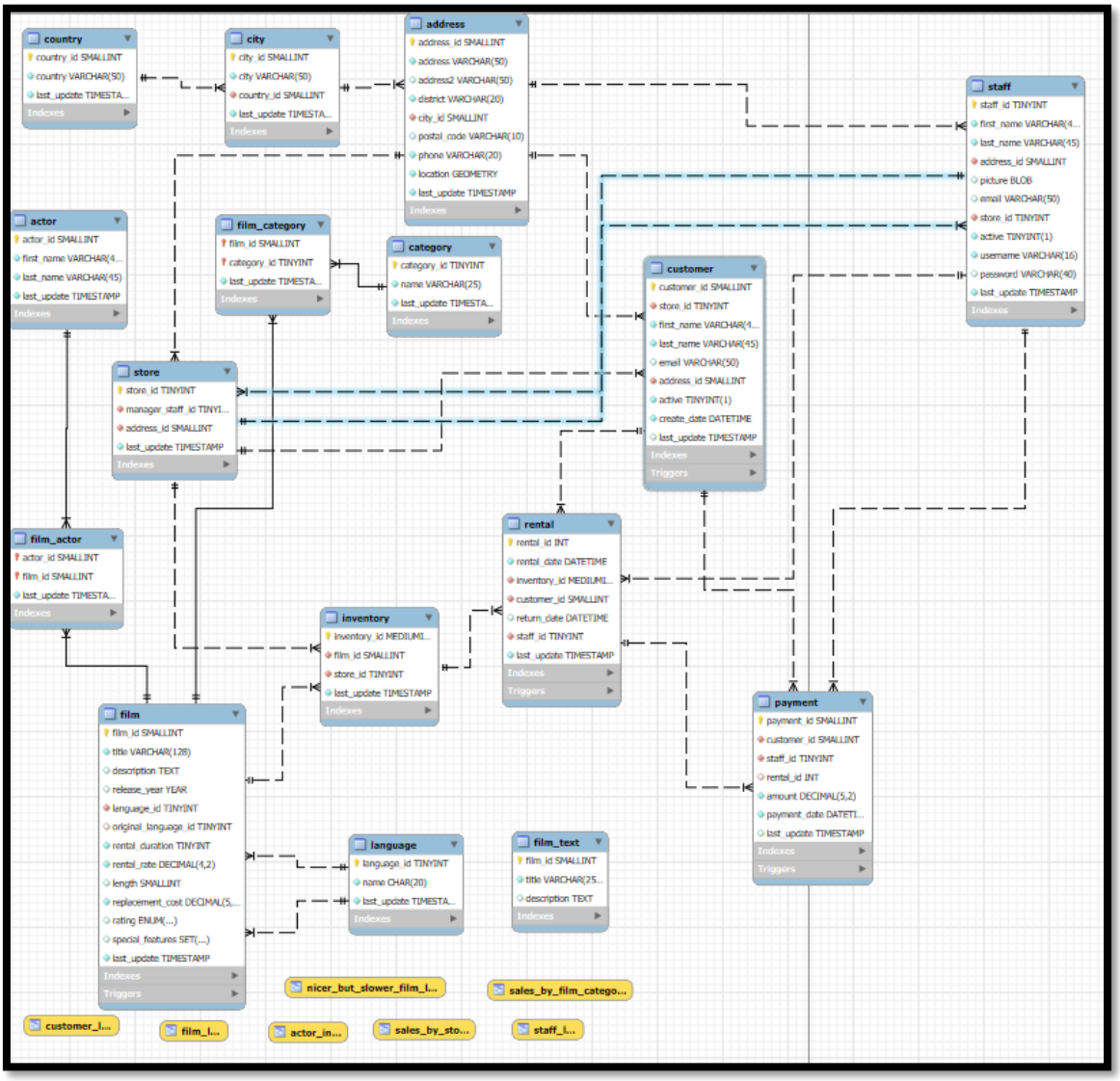
   -- a) Se seleccionan los campos 'first_name' y 'last_name' de la tabla 'customer'.

   -- b) El WHERE filtra aquellos clientes cuyo campo 'active' es igual a 1.

   SELECT first_name, last_name FROM customer WHERE active = 1;

   Antes de continuar con las consultas, vamos a hacer el DER de Sakila
   El modelo de datos de Sakila representa una tienda de alquiler de películas, y su estructura incluye varias tablas con relaciones bien definidas entre ellas, lo que permite crear un DER que visualice cómo se interrelacionan las entidades dentro de la base de datos.

   En el DER de Sakila, vamos a observar y comprender:



- **Relaciones claras y múltiples**: Sakila tiene relaciones entre diversas tablas como film, category, actor, customer, rental, payment, etc. Un DER te permite visualizar cómo estas tablas interactúan entre sí y cómo se organizan los datos.

- **Mejora de la comprensión de la base de datos**: Un DER puede ayudar a entender la estructura de la base de datos y cómo los datos fluyen entre diferentes tablas. Es útil para estudiantes o desarrolladores que quieren comprender el modelo de datos antes de realizar consultas complejas.

- **Optimización y diseño**: El DER puede ayudarte a identificar ***redundancias o mejorar la normalización*** de la base de datos. También facilita la tarea de ***diseñar y modificar*** la base de datos en futuras implementaciones.

- **Facilidad para la enseñanza y aprendizaje**: crear un DER facilita la enseñanza de conceptos como relaciones uno a muchos, muchos a muchos, claves primarias, claves foráneas, entre otros.

- **Documentación técnica**: El DER es también una herramienta útil en la documentación técnica de proyectos, especialmente cuando se tiene que presentar o modificar una base de datos a otros miembros del equipo de desarrollo.

3. **Mostrar las películas de categoría ‘Action’**:
   Buscará los títulos de las películas que pertenecen a la categoría pedida, en este caso Action.

```
-- 3. Películas de categoría 'Action'

-- Esta consulta busca todas las películas que pertenecen a la categoría 'Action'.

-- Paso a paso:

-- a) Se selecciona el título de las películas desde la tabla 'film' con el alias 'f'. Colocar ‘f’ como alias de film es como colocar film.title.

-- ‘f’, ‘fc’, ‘c’ son alias para simplificar las consultas y mejorar la legibilidad del código.

-- b) Se hace un INNER JOIN con la tabla 'film_category' para relacionar las películas con sus categorías a través del campo 'film_id'.

-- c) Se hace otro INNER JOIN con la tabla 'category' para obtener el nombre de la categoría mediante 'category_id'.

-- d) En el WHERE, se filtran las películas que tienen la categoría 'Action'.

SELECT f.title

FROM film f

INNER JOIN film_category fc ON f.film_id = fc.film_id

INNER JOIN category c ON fc.category_id = c.category_id

WHERE c.name = 'Action';

SELECT f.title FROM film f

INNER JOIN film_category fc ON f.film_id = fc.film_id

INNER JOIN category c ON fc.category_id = c.category_id

WHERE c.name = 'Action';
```

Sin usar el Alias

```
SELECT film.title

FROM film
```

```
INNER JOIN film_category ON film.film_id =
film_category.film_id

INNER JOIN category ON film_category.category_id =
category.category_id

WHERE category.name = 'Action';
```

Aclaración:

1- FROM film f:

Seleccionamos la tabla film, y le damos el alias f para simplificar las referencias en la consulta.

2- INNER JOIN film_category fc ON f.film_id = fc.film_id:

Hacemos un INNER JOIN entre la tabla film y la tabla film_category. Unimos ambas tablas utilizando el campo film_id. El alias fc es para la tabla film_category.

3- INNER JOIN category c ON fc.category_id = c.category_id:

Hacemos otro INNER JOIN, esta vez entre la tabla film_category y la tabla category. Unimos ambas tablas utilizando el campo category_id. El alias c es para la tabla category.

4- WHERE c.name = 'Action':

Filtramos el resultado para que solo aparezcan las películas que están en la categoría 'Action'.

5- ¿Por qué usamos INNER JOIN?

- **INNER JOIN** asegura que solo las películas que tengan una categoría asociada de tipo **'Action'** se incluyan en los resultados.

- Si una película no tiene un registro correspondiente en film_category o si la categoría no es **'Action'**, esa película no aparecerá en el resultado.

4. **Listar actores por orden alfabético**:
   Ordena los actores por apellido de forma ascendente.

```
-- 4. Listado de actores por orden alfabético

-- Esta consulta lista a los actores en orden alfabético por apellido.

-- a) Se seleccionan los nombres y apellidos de la tabla 'actor'.

-- b) Se ordenan los resultados en orden ascendente por el campo 'last_name'.

SELECT first_name, last_name FROM actor ORDER BY last_name ASC;
```

B. Nivel Intermedio

1. **Listar las películas alquiladas por clientes:**
   Muestra el nombre del cliente, el título de la película y la fecha del alquiler.

```
-- 1. Películas alquiladas por cliente

-- Esta consulta muestra las películas alquiladas por cada cliente.

-- a) Se seleccionan los nombres del cliente, el título de la película y la fecha de alquiler.

-- b) Se conectan las tablas rental, customer, inventory y film a través de sus IDs relacionados.

SELECT c.first_name, c.last_name, f.title, r.rental_date

FROM rental r

INNER JOIN customer c ON r.customer_id = c.customer_id

INNER JOIN inventory i ON r.inventory_id = i.inventory_id

INNER JOIN film f ON i.film_id = f.film_id;
```

2. **Películas sin alquiler en los últimos 3 meses:**
   Identifica las películas que no han sido alquiladas en los últimos 90 días.

```
-- 2. Películas sin alquiler en los últimos 3 meses

-- Esta consulta identifica las películas que no han sido alquiladas en los últimos 90 días.

-- a) Se utiliza LEFT JOIN para incluir todas las películas, incluso si no tienen alquileres recientes.
```

```
-- b) Se aplica una condición en el WHERE para verificar películas con alquileres nulos o anteriores a 90 días.

SELECT f.title

FROM film f

LEFT JOIN inventory i ON f.film_id = i.film_id

LEFT JOIN rental r ON i.inventory_id = r.inventory_id

WHERE r.rental_date IS NULL OR r.rental_date < NOW() - INTERVAL 90 DAY;
```

3. **Top o ranking de 5 clientes por cantidad de alquileres:**
   Muestra el listado de los 5 clientes que han alquilado más películas, incluyendo el total de alquileres.

```
-- 3. Top 5 clientes por cantidad de alquileres

-- Esta consulta muestra los 5 clientes que más películas han alquilado.

-- a) Se utiliza COUNT() para contar los alquileres por cliente.

-- b) Se agrupan los resultados por cliente y se ordenan en orden descendente.

-- c) Se limita el resultado a los 5 primeros clientes.

SELECT c.first_name, c.last_name, COUNT(r.rental_id) AS total_rentals

FROM rental r

INNER JOIN customer c ON r.customer_id = c.customer_id

GROUP BY c.customer_id

ORDER BY total_rentals DESC

LIMIT 5;
```

4. **Ingresos por categoría:**
   Calcula el total recaudado por cada categoría de película.

```
-- 4. Ingresos por categoría

-- Esta consulta calcula los ingresos totales por categoría de película.

-- a) Se suma el monto de cada pago utilizando SUM().

-- b) Se agrupan los resultados por categoría, identificando cada categoría mediante su nombre.

SELECT c.name AS category, SUM(p.amount) AS total_revenue
```

```
FROM payment p

INNER JOIN rental r ON p.rental_id = r.rental_id

INNER JOIN inventory i ON r.inventory_id = i.inventory_id

INNER JOIN film f ON i.film_id = f.film_id

INNER JOIN film_category fc ON f.film_id = fc.film_id

INNER JOIN category c ON fc.category_id = c.category_id

GROUP BY c.category_id;
```

5. **Clientes con más de 3 alquileres pendientes**
   Son los clientes que no han devuelto más de 3 películas alquiladas.

```
-- 5. Clientes con más de 3 alquileres pendientes

-- Esta consulta utiliza COUNT() para contar los alquileres pendientes (donde return_date es NULL) y HAVING para filtrar los clientes con más de 3 alquileres pendientes.

SELECT c.first_name, c.last_name, COUNT(r.rental_id) AS pending_rentals

FROM rental r

INNER JOIN customer c ON r.customer_id = c.customer_id

WHERE r.return_date IS NULL

GROUP BY c.customer_id

HAVING pending_rentals > 3;
```

6. **Comparar ingresos por ciudad**
   Compara los ingresos generados en cada tienda, mostrando la ciudad correspondiente.

```
-- 6. Comparar ingresos por ciudad

-- Esta consulta agrupa por ciudad utilizando la tabla address y calcula el total de ingresos por ciudad mediante SUM() aplicado sobre los montos de los pagos.

SELECT a.city, SUM(p.amount) AS total_revenue

FROM payment p

INNER JOIN rental r ON p.rental_id = r.rental_id

INNER JOIN customer c ON r.customer_id = c.customer_id

INNER JOIN address a ON c.address_id = a.address_id

GROUP BY a.city;
```

7. **Clientes con películas alquiladas, pendientes de devolver**
Muestra los clientes que actualmente tienen películas alquiladas, indicando los títulos y las fechas estimadas de devolución.

```
-- 7. Clientes con películas en stock

-- Esta consulta identifica a los clientes que aún tienen películas sin devolver (return_date es NULL) y muestra los títulos y fechas estimadas de devolución.

SELECT c.first_name, c.last_name, f.title, r.return_date

FROM rental r

INNER JOIN customer c ON r.customer_id = c.customer_id

INNER JOIN inventory i ON r.inventory_id = i.inventory_id

INNER JOIN film f ON i.film_id = f.film_id

WHERE r.return_date IS NULL;
```

Más ejercicios para descubrir Sakila

1. Listar películas por rango de duración de alquiler:
Queremos listar todas las películas cuyo rental_duration esté entre 5 y 10 días. Mostramos el título, año y duración de alquiler.

```
SELECT title, release_year, rental_duration
FROM film
WHERE rental_duration BETWEEN 5 AND 10;
```

2. Obtener clientes que han alquilado en una tienda específica:
Listar todos los clientes que han alquilado películas en una tienda específica (por ejemplo, en la ciudad de 'San Francisco').

```
SELECT DISTINCT c.first_name, c.last_name
FROM customer c
INNER JOIN address a ON c.address_id = a.address_id
INNER JOIN city ci ON a.city_id = ci.city_id
INNER JOIN rental r ON c.customer_id = r.customer_id
WHERE ci.city = 'San Francisco';
```

3. Películas con más de 3 categorías:
Queremos listar los títulos de las películas que están asociadas con más de tres categorías diferentes. Mostramos el título y el número de categorías asociadas.

```
SELECT f.title, COUNT(fc.category_id) AS category_count
FROM film f
INNER JOIN film_category fc ON f.film_id = fc.film_id
```

```
GROUP BY f.film_id
HAVING COUNT(fc.category_id) > 3;
```

4. Ranking de las películas más alquiladas:
Muestra el ranking de las 5 películas más alquiladas, con la cantidad de veces que se han alquilado.

```
SELECT f.title, COUNT(r.rental_id) AS rental_count
FROM rental r
INNER JOIN inventory i ON r.inventory_id = i.inventory_id
INNER JOIN film f ON i.film_id = f.film_id
GROUP BY f.film_id
ORDER BY rental_count DESC
LIMIT 5;
```

5. Listar actores que no han trabajado en ninguna película:
Identificar los actores que no tienen ninguna película asociada (en caso de que haya actores en la base de datos sin ninguna película asignada).

```
SELECT a.first_name, a.last_name
FROM actor a
LEFT JOIN film_actor fa ON a.actor_id = fa.actor_id
WHERE fa.film_id IS NULL;
```

6. Total de alquileres por día de la semana:
Obtenemos el total de alquileres realizados en cada día de la semana (por ejemplo, lunes, martes, etc.). Mostramos el día y el total de alquileres realizados en ese día.

```
SELECT DAYNAME(r.rental_date) AS weekday, COUNT(r.rental_id) AS
rental_count
FROM rental r
GROUP BY weekday
ORDER BY rental_count DESC;
```

7. Mostrar películas que no han sido alquiladas en todo el mes de abril de 2005:
Identificamos las películas que no fueron alquiladas durante todo el mes de abril de 2005.

```
SELECT f.title
FROM film f
LEFT JOIN inventory i ON f.film_id = i.film_id
LEFT JOIN rental r ON i.inventory_id = r.inventory_id
WHERE (r.rental_date IS NULL OR MONTH(r.rental_date) != 4 OR
YEAR(r.rental_date) != 2005);
```

8. Listar las películas alquiladas por clientes que viven en una ciudad específica:
Queremos listar las películas que han sido alquiladas por clientes que viven en una ciudad como 'Los Angeles'. Mostramos el título de la película, el nombre del cliente y la fecha de alquiler.

```
SELECT f.title, c.first_name, c.last_name, r.rental_date
FROM rental r
INNER JOIN customer c ON r.customer_id = c.customer_id
INNER JOIN address a ON c.address_id = a.address_id
INNER JOIN city ci ON a.city_id = ci.city_id
INNER JOIN inventory i ON r.inventory_id = i.inventory_id
INNER JOIN film f ON i.film_id = f.film_id
WHERE ci.city = 'Los Angeles';
```

9. Ver ingresos de clientes por ciudad y categoría:
Queremos calcular el total de ingresos generados por los clientes en cada ciudad, por categoría de película. Mostramos ciudad, categoría y total de ingresos.

```
SELECT ci.city, c.name AS category, SUM(p.amount) AS total_revenue
FROM payment p
INNER JOIN rental r ON p.rental_id = r.rental_id
INNER JOIN inventory i ON r.inventory_id = i.inventory_id
INNER JOIN film f ON i.film_id = f.film_id
INNER JOIN film_category fc ON f.film_id = fc.film_id
INNER JOIN category c ON fc.category_id = c.category_id
INNER JOIN customer cu ON r.customer_id = cu.customer_id
INNER JOIN address a ON cu.address_id = a.address_id
INNER JOIN city ci ON a.city_id = ci.city_id
GROUP BY ci.city, c.category_id
ORDER BY total_revenue DESC;
```

10. Ver las películas más recientes por cada actor:
Listar las películas más recientes en las que ha trabajado cada actor. Mostramos el nombre del actor, el título de la película y el año de lanzamiento.

```
SELECT a.first_name, a.last_name, f.title, f.release_year
FROM actor a
INNER JOIN film_actor fa ON a.actor_id = fa.actor_id
INNER JOIN film f ON fa.film_id = f.film_id
WHERE f.release_year = (SELECT MAX(release_year)
                        FROM film
                        WHERE film_id = f.film_id);
```