03. Consultas a Bases de Datos Estructuradas - Desafío 1

Se pide analizar la siguiente consulta SQL y explicar lo que se obtiene con la misma:

```sql
1 SELECT
2   fecha_renta,
3   titulo,
4   genero,
5   cantidad_pagada,
6 RANK() OVER(PARTITION BY genero ORDER BY cantidad_pagada DESC)
7 FROM pelicula
8 JOIN alquiler
9   ON alquiler.pelicula_id = pelicula.id;
```

Figura 1 - Consulta para analizar.

Esta consulta SQL mostrada en la Figura 1 va a mostrar una tabla que contendrá información de la fecha del alquiler de la película, el título, el género, y la cantidad de plata que se pagó por el alquiler de esa película (líneas de código 1, 2, 3, 4 y 5). Esta información está distribuida en dos tablas, por ejemplo la del título de la película y el género está en la tabla de películas, mientras que la información de la fecha de la renta y la cantidad pagada están en la tabla de alquiler. Es por este motivo que se deben combinar estas tablas con una cláusula JOIN, y se combinan a partir de un valor en común de ambas tablas que es el identificador de la película (líneas de código 8 y 9). Además se agrega una columna de información, donde se genera un ranking considerando la cantidad pagada, ordenado de mayor a menor, y separado por género de película (línea de código 6). Es decir, la consulta genera una tabla con las columnas “fecha_renta”, “titulo”, “genero”, “cantidad_pagada” y “ranking”. El ranking se reinicia para cada género, asignando el valor 1 a la película que generó la mayor cantidad de ingresos por alquiler dentro de su género. Los valores enteros [1, 2, 3, 4, 5 ...] indican la posición de cada película en función de los ingresos, donde un número menor significa que la película recaudó más que las siguientes en la lista.

Para corroborar lo esperado se generaron tablas de películas y alquiler en un editor online de SQL , se ingresaron algunos registros y se realizó la consulta. Esto se muestra en la Figura 2.

fecha_renta	titulo	genero	cantidad_pagada	RANK() OVER(PARTITION BY ...
2024-08-20	Mad Max: Fury Road	Acción	13.00	1
2024-08-17	The Dark Knight	Acción	10.00	2
2024-08-18	Interstellar	Ciencia Ficción	14.00	1
2024-08-15	Inception	Ciencia Ficción	12.00	2
2024-08-19	The Godfather	Drama	18.00	1
2024-08-16	Titanic	Drama	15.00	2

Figura 2 - Tabla generada a partir de la consulta dada.

Esto es lo que se obtuvo de la consulta tal y como se esperaba.

Una forma de que esta consulta quede más prolija es escribir un alias para el ranking, poniendo al final de la línea 6 “AS ranking”, y ahí quedaría esa columna de la consulta con este nombre como se muestra en la Figura 3.

fecha_renta	titulo	genero	cantidad_pagada	ranking
2024-08-20	Mad Max: Fury R...	Acción	13.00	1
2024-08-17	The Dark Knight	Acción	10.00	2
2024-08-18	Interstellar	Ciencia Ficción	14.00	1
2024-08-15	Inception	Ciencia Ficción	12.00	2
2024-08-19	The Godfather	Drama	18.00	1
2024-08-16	Titanic	Drama	15.00	2

Figura 3 - Tabla generada con alias.

Para la creación de las tablas y el ingreso de registros se generó este código:

```
1 CREATE TABLE pelicula (
2   id int PRIMARY KEY,
3   titulo varchar(255),
4   genero varchar(255)
5 );
6
7 INSERT INTO pelicula (id, titulo, genero) VALUES
8 (1, 'Inception', 'Ciencia Ficción'),
9 (2, 'Titanic', 'Drama'),
10 (3, 'The Dark Knight', 'Acción'),
11 (4, 'Interstellar', 'Ciencia Ficción'),
12 (5, 'The Godfather', 'Drama'),
13 (6, 'Mad Max: Fury Road', 'Acción');
```

```sql
CREATE TABLE alquiler (
  id int PRIMARY KEY,
  pelicula_id int,
  fecha_renta date,
  cantidad_pagada varchar(255),
  FOREIGN KEY (pelicula_id) REFERENCES pelicula(id)
);

INSERT INTO alquiler (id, pelicula_id, fecha_renta, cantidad_pagada) VALUES
(1, 1, '2024-08-15', '12.00'),
(2, 2, '2024-08-16', '15.00'),
(3, 3, '2024-08-17', '10.00'),
(4, 4, '2024-08-18', '14.00'),
(5, 5, '2024-08-19', '18.00'),
(6, 6, '2024-08-20', '13.00');
```

Una vez analizada la consulta se pide diseñar una tabla de **Clientes** con al menos 3 campos y armar una nueva consulta que incorpore datos de estos, ordenados por nombre de película y cliente.

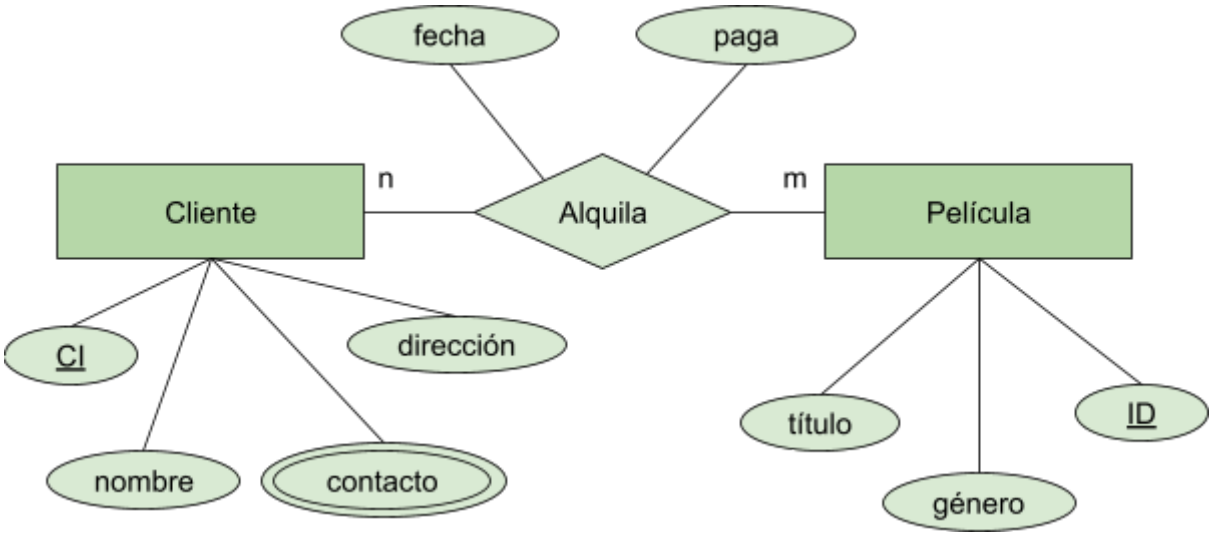
Al crear la tabla de clientes, se ha decidido utilizar la cédula de identidad como clave primaria para identificar de manera única a cada cliente. Además, se almacenarán su nombre y apellido, su dirección, y al menos dos formas de contacto, como su dirección de correo electrónico y su número de celular. Es fundamental registrar tanto la dirección (que podría presentarse en una factura emitida a su nombre) como los datos de contacto, ya que al alquilarle una película, contar con esta información proporciona mayor seguridad en caso de necesitar contactarlo para asegurar la devolución del material alquilado.

```sql
CREATE TABLE clientes (
    ci varchar(255) PRIMARY KEY,
    nombre varchar(255),
    apellido varchar(255),
    direccion varchar(255)
);

```

Dado que un cliente puede tener múltiples formas de contacto (atributo multivaluado.), y para mantener una estructura de base de datos normalizada, se crea una tabla separada llamada Contacto_Cliente. En esta tabla se relaciona cada contacto con la cédula del cliente a través de una clave foránea (ci_cliente), permitiendo así almacenar múltiples contactos asociados a un mismo cliente.

```sql
CREATE TABLE Contacto_Cliente (
    ci_cliente varchar(255),
    contacto,
    FOREIGN KEY (ci_cliente) REFERENCES clientes(ci)
);

```

Respecto a los clientes se podría hacer varias consultas, por ejemplo si la tabla de ALQUILER estuviera relacionada con cliente se podría generar una consulta para ver qué películas alquiló cada uno de los clientes, incluso consultar por un cliente particular. Un modelo posible para modelar esta realidad podría ser el siguiente, donde se consideran las entidades de cliente y película, con sus atributos ya mencionados, y se relacionan a través de alquiler.



Una consulta posible podría ser la siguiente:

```sql
-- Qué películas alquiló cada uno de los clientes y en que fecha
-- ordenado por titulo de pelicula y nombre de cliente
SELECT
    c.nombre,
    p.titulo,
    a.fecha
FROM cliente c
JOIN
    alquiler a ON c.ci = a.ci_cliente
JOIN
    pelicula p ON p.id = a.id_pelicula
ORDER BY
    p.titulo ASC,
    c.nombre ASC;
```