DQL CONSULTAS

Operadores Aritméticos

1. **Multiplicación**

Consulta: Devuelve el número de butacas y su valor multiplicado por 2

```
SELECT butacas, butacas * 2 AS "Doble de butacas"
FROM salas;
```

2. **Suma**

Consulta: Incrementa en 1 el precio de los tickets

```
SELECT precio, precio + 1 AS "Precio Incrementado"
FROM tickets;
```

3. **Modulo**

Consulta: Para saber si el número de butacas es par o impar

```
SELECT butacas,
    CASE
        WHEN butacas % 2 = 0 THEN 'PAR'
        ELSE 'IMPAR'
    END AS "Tipo Butacas"
FROM salas;
```

4. **División**

Consulta: Calcular la mitad de las butacas disponibles por sala

```
SELECT idSala, butacas, butacas / 2 AS "Mitad Butacas"
FROM salas;
```

Operadores de comparación unitarios

Uso de la cláusula WHERE

1.Consulta: Selecciona los nombres de cines que están en Valencia

```
SELECT nombre
FROM cines
WHERE ciudad =  'Valencia';
```

2.Consulta: para seleccionar los precios de los productos que no cuesten 50 euros:

```
SELECT precio
FROM productos
WHERE precio != 50;
```

3.Consulta: Selecciona las salas con más de 110 butacas

```
SELECT idSala, butacas
FROM salas
WHERE butacas > 110;
```

4.Consulta: Busca la sala con exactamente 111 butacas

```
SELECT idSala, butacas
FROM salas
WHERE butacas = 111;
```

5.Consulta: Muestra los cines cuyo ID sea menor a 5

```
SELECT idCine, nombre
FROM cines
WHERE idCine < 5;
```

6.Consulta: Encuentra los cines con IDs mayores o iguales a 40

```
SELECT idCine, nombre
FROM cines
WHERE idCine >= 40;
```

Operadores de comparación sobre valores individuales

Uso de IS NULL

Consulta: para comprobar si un valor esta vacío

```
SELECT * FROM Clientes WHERE email IS NULL;
```

Operadores Lógicos

1.AND

Consulta: seleccionar cines en Valencia con un ID mayor a 10.

```
SELECT nombre, idCine
FROM cines
WHERE ciudad = 'Valencia' AND idCine > 10;
```

2.NOT

Consulta:Excluir los cines que están en Madrid.

```
SELECT nombre, ciudad
FROM cines
WHERE NOT ciudad = 'Madrid';
```

3.OR

Consulta: Seleccionar cines cuyo precio sea menor que 5 o mayor que 20 euros.

```
SELECT nombre, precio
FROM cines
WHERE precio < 5 OR precio > 20;
```

4. NOT IN

Consulta: Seleccionar los cines que no están en las ciudades 'Valencia' y 'Madrid'.

```
SELECT nombre, ciudad
FROM cines
WHERE ciudad NOT IN ('Valencia', 'Madrid');
```

Operadores Comodines (_ y %)

Uso de LIKE

1.Consulta: encuentra cines cuyos nombres empiezan con "C".

```
SELECT nombre
FROM cines
WHERE nombre LIKE 'C%';
```

2.Consulta: Encuentra poblaciones cuyos nombres terminan en "a".

```
SELECT nombre
FROM poblaciones
WHERE nombre LIKE '%a';
```

3.Consulta: Busca cines con un nombre que tenga "ol" como segundo y tercer carácter.

```
SELECT nombre
FROM cines
WHERE nombre LIKE '_ol%';
```

4. Consulta:busca poblaciones que empiezan con "A" y terminan con "a".

```
SELECT nombre
FROM poblaciones
WHERE nombre LIKE 'A%a';
```

5.Busca cines cuyo nombre contiene "cine".

```
SELECT nombre
FROM cines
WHERE nombre LIKE '%cine%';
```

Funciones sobre cadenas de texto

1.CONCAT()

Consulta:muestra los nombres de los cines junto con la localidad en un solo campo.

```
SELECT CONCAT(nombre, ' - ', localidad) AS cine_localidad
FROM cines;
```

2. LEFT

Consulta:muestra los tres primeros caracteres del nombre de cada cine.

```
SELECT nombre, LEFT(nombre, 3) AS primeros_tres
FROM cines;
```

3. LOWER/UPPER

Consulta:muestra los nombres de los cines en minúsculas y en mayúsculas.

```
SELECT nombre, LOWER(nombre) AS en_minusculas, UPPER(nombre) AS en_mayusculas
FROM cines;
```

Funciones sobre valores numericos

1. **ROUND**

Consulta: muestra el precio de los tickets redondeado a 2 decimales.

```
SELECT precio, ROUND(precio, 2) AS precio_redondeado
FROM tickets;
```

Funciones agregadas

1. **COUNT()**

Consulta:cuenta cuántos tickets existen en la tabla.

```
SELECT COUNT(*) AS total_tickets
FROM tickets;
```

2. **SUM()**

Consulta: calcula el total de ingresos sumando los precios de los tickets vendidos.

SELECT SUM(precio) AS total_ingresos

FROM tickets;

3. **AVG**

Consulta: muestra el precio promedio de los tickets

SELECT AVG(precio) AS precio_promedio

FROM tickets;

4. **MAX()**

Consulta: encuentra el ticket más caro.

SELECT MAX(precio) AS precio_maximo

FROM tickets;

5. **MIN()**

Consulta: encuentra el ticket con el precio más barato.

SELECT MIN(precio) AS precio_minimo

FROM tickets;

Consultas con valores de resumen

1. **GROUP BY**

Consulta: muestra cuántas salas tiene cada cine

```
SELECT id_cine, COUNT(*) AS total_salas
FROM salas
GROUP BY id_cine;
```

2. **ORDER BY**

Consulta: ordena los cines por el precio más alto de sus tickets, de menor a mayor.

```
SELECT id_cine, MAX(precio) AS precio_maximo
FROM tickets
GROUP BY id_cine
ORDER BY precio_maximo ASC;
```

3. **HAVING**

Consulta: encuentra los cines donde el precio medio del ticket es mayor a 10€.

```
SELECT id_cine, AVG(precio) AS precio_medio
FROM tickets
GROUP BY id_cine
HAVING AVG(precio) > 10;
```

Tipos de JOINS

Siguiendo estas tablas:

```
CREATE TABLE clientes (
   id_cliente INT PRIMARY KEY,
   nombre VARCHAR(50)
);

CREATE TABLE pedidos (
   id_pedido INT PRIMARY KEY,
   id_cliente INT,
   producto VARCHAR(50),
   FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

1. **INNER JOIN**

Consulta: encontrar todos los pedidos con el nombre del cliente que los hizo

```
SELECT clientes.nombre, pedidos.producto
FROM clientes
INNER JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

INNER JOIN: (Solo los que tienen coincidencia en ambas tablas)

2. **LEFT JOIN**

Consulta: mostrar todos los clientes, incluyendo aquellos que no han hecho pedidos

```
SELECT clientes.nombre, pedidos.producto
FROM clientes
LEFT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

LEFT JOIN (Todos los clientes, aunque no tengan pedidos)

3. **RIGHT JOIN**

Consulta: mostrar todos los pedidos, aunque el cliente no exista

```
SELECT clientes.nombre, pedidos.producto
FROM clientes
RIGHT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

RIGHT JOIN asegura que todos los pedidos aparezcan, aunque el cliente no esté en la tabla clientes.

4. **JOIN**

Consulta: listar los productos pedidos por "Ana"

```
SELECT pedidos.producto
FROM pedidos
JOIN clientes ON clientes.id_cliente = pedidos.id_cliente
WHERE clientes.nombre = 'Ana';
```

JOIN une las tablas para obtener los pedidos de Ana.

Subconsultas

Siguiendo estas tablas:

```
CREATE TABLE clientes (
   id_cliente INT PRIMARY KEY,
   nombre VARCHAR(50)
);

CREATE TABLE pedidos (
   id_pedido INT PRIMARY KEY,
   id_cliente INT,
   producto VARCHAR(50),
   FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)
);
```

1.IN

Consulta: Encontrar clientes que han hecho pedidos.

```
SELECT nombre
FROM clientes
WHERE id_cliente IN (SELECT id_cliente FROM pedidos);
```

2. ANY

Consulta: Encontrar clientes con id_cliente mayor que algún cliente que ha hecho pedidos.

```
SELECT nombre
FROM clientes
WHERE id_cliente > ANY (SELECT id_cliente FROM pedidos);
```

3. ALL

Consulta: Encontrar clientes con id_cliente mayor que todos los clientes que han hecho pedidos.

```
SELECT nombre
FROM clientes
WHERE id_cliente > ALL (SELECT id_cliente FROM pedidos);
```

4. EXIST

Consulta: Encontrar clientes que tienen al menos un pedido.

```
SELECT nombre

FROM clientes c

WHERE EXISTS (SELECT 1 FROM pedidos p WHERE c.id_cliente = p.id_cliente);
```