Trabalho SQL - Querys

P1) Quais os títulos, dias e horas dos espetáculos registados na BD?

SELECT titulo,dia,hora
FROM public.espetaculo;

- Esta é uma query simples, que responde ao objectivo e mantém a coluna dia no seu formato padrão do banco de dados.

Algumas outras querys apresentadas:

SELECT titulo, to_char(dia,'DD/MM/YYYY'), hora
FROM public.espetaculo;

- Esta query também está correcta e transforma a coluna dia em um formato de data legível (DD/MM/YYYY) usando to_char().

Outra query apresentada:

SELECT DISTINCT ON (titulo) titulo, dia, hora
FROM public."espetaculo"
Order by 1,2;
Esta query usa o DISTINCT ON (titulo) sem uma ordenação clara

- DISTINCT ON (titulo) retorna apenas uma linha para cada valor único de titulo, mas a escolha de qual linha manter depende da cláusula ORDER BY.
- ORDER BY 1,2 (equivalente a ORDER BY titulo, dia) não garante qual dia e hora serão escolhidos para cada título.
- DISTINCT ON (titulo) não responde à questão porque não apresenta todos os dias e horas dos espetáculos registados na BD

Se o objetivo é simplesmente listar todos os títulos, dias e horas dos espetáculos registados na base de dados, então DISTINCT ON (titulo) não é necessário. O DISTINCT simples já garante que não haja repetições exatas de linhas.

P2) Quais as categorias e os títulos dos espetáculos registados na BD?

select titulo, categoria from public.espetaculo;
OU
select DISTINCT titulo, categoria from public.espetaculo;

Diferença entre querys::
Ambas as querys estão correctas no entanto usamos de acordo com o nosso objectivo:

- Se a BD tiver registros duplicados e queremos evitar repetições: Usamos SELECT DISTINCT.
- Se queremos ver todos os registros, incluindo repetições: Usamos SELECT sem DISTINCT.

Neste caso a primeira query mostra todos os registos e como ainda estamos numa fase inicial de análise poderá ser relevante ver todos os registos e ter uma visão completa da BD ao nível de categorias e títulos.
No entanto a segunda query remove os duplicados de forma direta, sendo mais eficiente, mais clara e fácil de entender a nível de output.

Outra query apresentada foi:
select titulo, categoria from public.espetaculo
group by categoria, titulo
order by titulo asc

Neste caso, e ainda que esta query retorne o mesmo resultado não é a forma mais indicada de resposta à questão. A utilização do group by é menos eficiente do que o distinct para eliminar duplicados (se for este o objectivo) e não havendo agregações o group by não acrescenta valor, se fosse para contar quantos espetáculos por categoria, ai sim o group by acrescentaria valor.

P3) Quais os nomes e datas de nascimento dos cantores registados? Ordene o resultado por data de nascimento crescente.

SELECT nome,d_nasc
FROM public.artista
WHERE tipo='cantor'
ORDER BY d_nasc ASC;

OU

SELECT nome,d_nasc
FROM public.artista
WHERE tipo like'cantor'
ORDER BY d_nasc ASC;

Diferença entre querys::
Ambas as querys estão correctas no entanto :

- A primeira query (WHERE tipo = 'cantor') é a mais correta e eficiente pois usa uma comparação directa e retorna apenas os registros onde o tipo é exatamente igual a cantor
- A segunda query usa LIKE mas sem %, logo age como um simples = , e normalmente LIKE é usado para buscas parciais como LIKE ‘cant%’ (para encontrar cantor, cantora, etc)

Muitos colocaram apenas a query abaixo, mas está **incorrecta** porque não está a filtrar por cantores, não utiliza a função where que é aqui a condição essencial da questão.

```
SELECT "nome", "d_nasc"
FROM artista
ORDER BY "d_nasc" ASC;
```

P4) Quais os emails e os nomes dos espectadores do Porto? Ordene o resultado por email.

```
SELECT email, nome
FROM public.espetador
WHERE cidade = 'Porto'
ORDER BY email ASC;
```

OU

```
SELECT email, nome
FROM public.espetador
WHERE cidade LIKE 'Porto'
ORDER BY email ASC;
```

Mesma lógica da query acima.

P5) Relativamente a cada artista, liste o seu nome e o dia e o título dos espetáculos em que foi o artista principal. Ordene o resultado por nome crescente e por data decrescente

```
SELECT a.nome, e.dia, e.titulo
FROM public.artista a
JOIN public.espetaculo e
ON a.nif=e.artista
ORDER BY a.nome ASC,
e.dia DESC;
```

Neste caso a não utilização do distinct permite mais uma vez ter uma visualização completa dos espetáculos porque os mesmos existem em diferentes horas. Apesar de a query não pedir horas a query acima (sem distinct) da-nos a visão completa de cada vez em que o artista foi o artista principal do espetáculo (mesmo que tenha acontecido mais do que um no mesmo dia, porque em diferentes horas)

P6) Relativamente a cada espetador, liste o seu nome e o lugar e o custo dos bilhetes que adquiriu. Ordene o resultado por nome crescente e por custo decrescente.

```
SELECT es.nome, b.lugar, b.custo
FROM public.espetador es
```

JOIN public.bilhete b
ON es.email = b.email
ORDER BY es.nome ASC,
b.custo DESC;

Também foi apresentada a query abaixo, mas não funciona e não está correcta porque a coluna email é ambígua. Falta especificar a relação correctamente, algo como join espetador on bilhete.email = espetador. email
O mesmo quando o id é a chave de ligação: id de qual tabela?

SELECT nome AS espectador, lugar, custo
FROM bilhete b
JOIN espetador ON email
ORDER BY nome ASC, custo DESC;

P7) Indique, sem repetições, o nome dos espectadores que compraram bilhetes para os espetáculos e o nome do artista.

SELECT DISTINCT e.nome as nome_espetador, a.nome as nome_artista
FROM public.espetador e
JOIN public.bilhete b ON e.email = b.email
JOIN public.espetaculo es ON b.id = es.id
JOIN public.artista a ON es.artista = a.nif;

A query abaixo também foi utilizada mas não está totalmente correcta, porque query não atende totalmente ao pedido, pois inclui o título do espetáculo, e DISTINCT ON pode eliminar associações entre artistas e espectadores.

SELECT distinct on (e.nome) e.nome, es.titulo, a.nome
FROM public.artista a
RIGHT JOIN public.Espetaculo es ON a.nif = es.artista
RIGHT JOIN public.Bilhete b ON es.id = b.id
LEFT JOIN public.Espetador e ON b.email = e.email
ORDER BY e.nome, es.artista, es.titulo;

P8) Liste os nomes de todas as pessoas, espectadores e artistas, desde que os espectadores sejam do Porto e os artistas sejam atores.

SELECT nome FROM espetador
WHERE cidade = 'Porto'
UNION ALL
SELECT nome FROM artista
WHERE tipo = 'ator';

OU

SELECT nome FROM espetador
WHERE cidade = 'Porto'
UNION

SELECT nome FROM artista
WHERE tipo = 'ator';

Diferença entre querys::
Ambas as querys dão o mesmo resultado (neste caso específico) no entanto :

- Usamos UNION ALL se quisermos ver todas as ocorrências, mesmo que haja repetições.
- Usamos UNION se desejamos listar apenas nomes únicos (evitando repetições).

Neste caso a segunda query seria a mais indicada, a intenção é ter uma lista única de nomes, logo UNION é a melhor opção.

P9) Liste os nomes de todas as pessoas que não são do Porto e que não têm nomes de artistas.

SELECT ep.nome FROM espetador ep
WHERE ep.cidade != 'Porto'
AND ep.nome NOT IN
(SELECT nome FROM artista);

OU

SELECT nome
FROM public.espetador
WHERE cidade!='Porto'
EXCEPT
SELECT nome
FROM public.artista;

Diferença entre querys::
Ambas as querys estão correctas, no entanto :
A segunda query (EXCEPT) é a melhor opção, pois:

- É mais eficiente em grandes volumes de dados.
- Evita problemas com valores NULL, porque no caso da primeira query se a subquery (SELECT nome FROM artista) retornar valores NULL, a condição NOT IN pode não funcionar corretamente e retornar um conjunto vazio)
- Trabalha diretamente com a diferença entre conjuntos, sem necessidade de verificar cada linha individualmente (o que acontece na primeira query que compara cada linha da tabela com a subconsulta
- resolve o problema de maneira mais simples e eficiente, listando corretamente as pessoas que não são do Porto e que não têm um nome igual ao de um artista

Alguns de vocês apresentaram query abaixo, pois a questão poderia levar a diferentes interpretações, ainda que o objectivo fosse a resposta acima.
Como existe uma Rita Almeida (espectadora) e uma Rita Pereira (atriz) para que os resultados retornem de maneira a que pelo menos um dos nomes do artista não seja o mesmo que do espectador, separaram os primeiros dos últimos nomes e assim garantem que nenhum nome era igual.

```
SELECT es.nome
FROM espectador es
WHERE
   (SPLIT_PART(es.nome, ' ', 1) NOT IN (
      SELECT SPLIT_PART(nome, ' ', 1) FROM artista
   ) AND
   SPLIT_PART(es.nome, ' ', 2) NOT IN (
      SELECT SPLIT_PART(nome, ' ', 2) FROM artista
   ))
AND es.cidade != 'Porto';
```

Outros apresentaram a query assim:

```
SELECT nome FROM public.espetador
WHERE cidade NOT LIKE 'Porto';
```

Esta query não está correcta, e apesar de dar o mesmo resultado (dado que a base é muito simples e pequena):

- Retorna os nomes dos espectadores cuja cidade não é "Porto".
- Problema: Não exclui os nomes que também aparecem na tabela artista, ou seja, pode incluir pessoas que compartilham o nome com um artista.
- Não atende totalmente ao pedido, pois apenas filtra pela cidade.

Esta foi outra query apresentada:

```
SELECT es.nome AS pessoa_nome FROM public.espetador es
WHERE es.cidade != 'Porto'
AND NOT EXISTS (SELECT 1 FROM public.artista a WHERE es.nome LIKE CONCAT('%', a.nome,
'%') OR es.nome LIKE CONCAT('%', SPLIT_PART(a.nome, ' ', 1), '%') OR es.nome LIKE
CONCAT('%', SPLIT_PART(a.nome, ' ', 2), '%') );
```

Filtra corretamente os espectadores que não são do Porto (WHERE es.cidade != 'Porto').
Usa NOT EXISTS para verificar se o nome do espectador contém um nome de artista.
Problema:

- A comparação de nomes tenta evitar coincidências parciais usando LIKE e SPLIT_PART, mas pode causar falsos positivos ou falsos negativos.
- Por exemplo, se um artista se chama "Carlos Silva" e um espectador for "Silvana", essa query pode eliminá-lo incorretamente.
- O uso de LIKE pode ser ineficiente em bases de dados grandes.

P10) Qual a receita total por espetáculo? Indique o id, o título e o valor total dos bilhetes

```
SELECT e.id, e.titulo, SUM(b.custo) AS receita_total
FROM espetaculo e
JOIN bilhete b ON e.id = b.id
GROUP BY e.id, e.titulo
order by id asc
```

Uma das querys apresentadas foi a que está abaixo, mas não está correcta porque:

```
select id, titulo, sum(preco) as receita_total
from public.espetaculo
group by id, titulo
```

order by id;

Está a tentar calcular a receita total somando preco da tabela espetaculo, mas:

- O preço correto a ser somado deveria ser o dos bilhetes vendidos, que está na tabela bilhete.
- A receita deve ser baseada no custo dos bilhetes comprados, e não apenas em um possível preço base do espetáculo.

Assim esta query não reflete corretamente a receita total, pois não considera os bilhetes vendidos.

P11) Qual a receita total por categoria de espetáculo? Indique a categoria e o valor total dos bilhetes.

SELECT e.categoria, SUM(b.custo) AS receita_total
FROM espetaculo e
JOIN bilhete b ON e.id = b.id
GROUP BY e.categoria;

Uma das querys apresentadas foi a que está abaixo, mas não está correcta porque:
Está a tentar somar o preço na tabela espetaculo, mas:

- O preço dos ingressos está na tabela bilhete, não em espetaculo.
- O valor total dos bilhetes deve ser calculado somando os custos dos bilhetes vendidos (b.custo), não um possível preço base do espetáculo.
- Não está considerando a relação entre bilhetes vendidos e espetáculos.
- Não reflete corretamente a receita total de bilhetes vendidos, pois usa a tabela errada.

```
select categoria, sum(preco) as receita_total
from public.espetaculo
group by categoria
order by receita_total;
```

P12) Quem (email) foi a todos os espetáculos do Tony Carreira?

Facilitava dividir a query em steps:

```
—1º verificar "email" que foi a pelo menos 1 espetaculo do Tony Carreira:
SELECT b.email
FROM bilhete b
JOIN espetaculo e ON b.id = e.id
JOIN artista a ON e.artista = a.nif
WHERE a.nome = 'Tony Carreira'
GROUP BY b.email
-- 2º verificar email que foi a todos os espetaculos do Tony Carreira (subquery):
HAVING COUNT(DISTINCT e.id) = (
        SELECT COUNT(*)
        FROM espetaculo e
        JOIN artista a ON e.artista = a.nif
        WHERE a.nome = 'Tony Carreira'
);
```

OU

```
SELECT b.email
FROM public.bilhete b
JOIN public.espetaculo esp ON b.espetaculo_id=esp.id
JOIN public.artista a ON a.nif=esp.artista
WHERE nome='Tony Carreira'
GROUP BY email
HAVING COUNT(DISTINCT b.espetaculo_id) = --COUNT(DISTINCT)-->Compara quantos
espetáculos do Tony Carreira cada espectador assistiu
   (SELECT COUNT(*) ---Conta quantos espetáculos do Tony Carreira existem
   FROM public.espetaculo esp
   JOIN public.artista a ON esp.artista = a.nif
   WHERE a.nome='Tony Carreira'); --Se os dois números calculados em cima forem iguais, significa
que a pessoa foi a todos.
OU usando CTE´s
-- Encontrar o NIF do Tony Carreira
WITH tony_carreira AS (
   SELECT nif
   FROM artista
   WHERE nome = 'Tony Carreira'
),
-- Encontrar todos os espetáculos do Tony Carreira
espetaculos_tony AS (
   SELECT id
   FROM espetaculo
   WHERE artista = (SELECT nif FROM tony_carreira)
),
-- Contar quantos espetáculos do Tony Carreira cada espetador assistiu
espetadores_tony AS (
   SELECT b.email, COUNT(DISTINCT b.id) AS num_espetaculos_assistidos
   FROM bilhete b
   JOIN espetaculos_tony et ON b.id = et.id
   GROUP BY b.email
)
-- Selecionar os espetadores que assistiram a todos os espetáculos do Tony Carreira
SELECT e.email
FROM espetador e
JOIN espetadores_tony et ON e.email = et.email
WHERE et.num_espetaculos_assistidos = (SELECT COUNT(*) FROM espetaculos_tony);
```

A query abaixo está incorrecta porque:

```
SELECT DISTINCT b.email
FROM bilhete b
JOIN espetaculo e ON b.id = e.id
JOIN artista a ON CAST(e.artista AS VARCHAR) = a.nif
WHERE a.nome = 'Tony Carreira';
```

- A query encontra os espectadores (emails) que assistiram aos espetáculos de Tony Carreira, mas não verifica se eles assistiram a todos os espetáculos dele.

- A questão P12 requer que a query identifique os emails dos espetadores que assistiram a todos os espetáculos de Tony Carreira.
- Para isso, é necessário comparar o número de espetáculos que o espectador assistiu com o número total de espetáculos de Tony Carreira. Essa lógica não está presente na query fornecida.

Houve ainda quem fizesse a correção da data nascimento do Tony Carreira :) com a query:

```
UPDATE artista
SET d_nasc = '1960-01-02'
WHERE a.nif = 1177289;
```

A criação das tabelas poderia ser feita com a importação dos csv´s ou com as querys CREATE TABLE e INSERT INTO.