

MÓDULO

BASE DE DADOS MYSQL

UNIDADE

MYSQL AVANÇADO

ÍNDICE

OBJETIVOS.....	3
INTRODUÇÃO.....	4
1. RECUPERAÇÃO DA CÓPIA DE SEGURANÇA.....	5
2. VIEWS	7
3. SUBCONSULTAS OU SUBQUERIES	13
CONCLUSÃO.....	21
AUTOAVALIAÇÃO	23
SOLUÇÕES.....	27
PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO.....	28
BIBLIOGRAFIA	29

OBJETIVOS

Com esta unidade didática, pretende-se que desenvolva os seguintes objetivos de aprendizagem:

- Aprender a recuperar backups.
- Saber criar e usar visualizações aproveitando todo o seu potencial.
- Saber como executar consultas agrupadas.

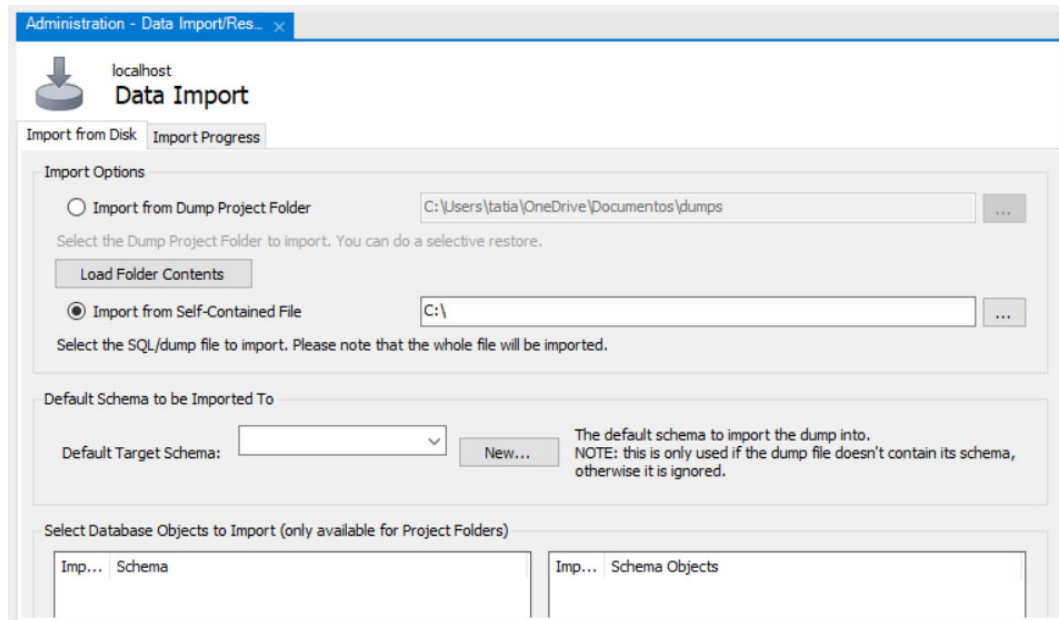
INTRODUÇÃO

Com esta unidade, aprenderá de uma forma muito básica e simples como fazer a diferença. Mas lembre-se de que, se quiser destacar-se, terá de continuar a ver tutoriais, ler livros e manuais, e consultar qualquer outra fonte de informação especializada que permita que siga em frente com o seu objetivo de ser especialista em bases de dados.

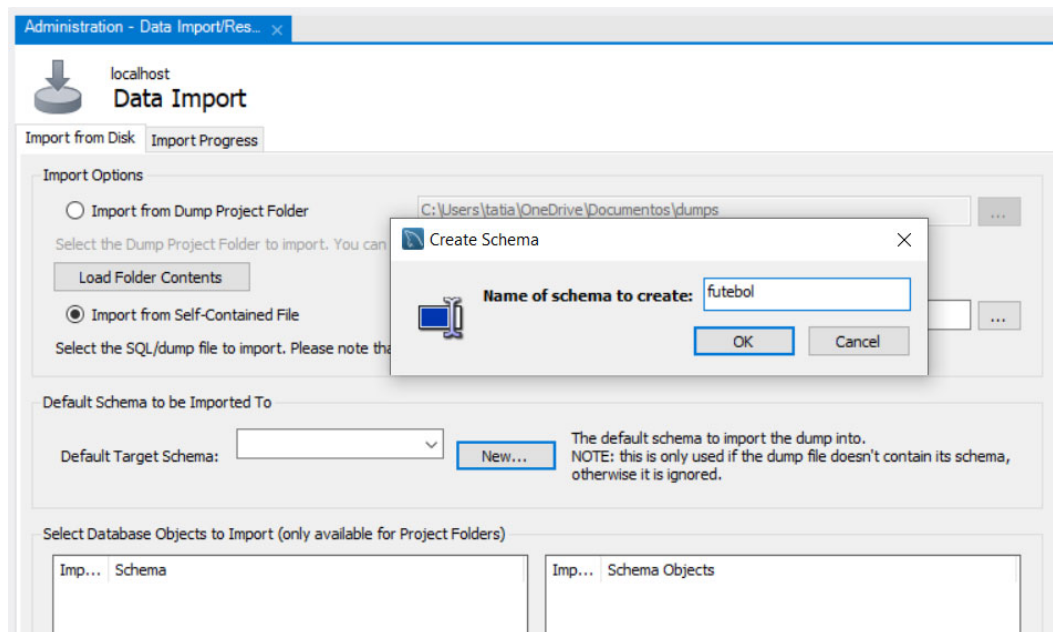
Deste modo, ao longo da unidade, aprenderá como recuperar um backup, assim como realizar consultas e visualizações agrupadas. No final, descobrirá a utilidade destas operações, que, neste momento, podem parecer algo estranhas.

1. RECUPERAÇÃO DA CÓPIA DE SEGURANÇA

A primeira etapa para recuperar um backup será abrir o MySQL Workbench e abrir a conexão com o utilizador “root”, como realizado anteriormente. Uma vez inserida a password, deverá aceder ao menu superior “Server” e, em seguida, a “Data Import”, onde aparecerá uma janela com várias opções.



A seguir, selecione a aba “Import from Disk” e escolha a opção “Import from Self-Contained File”, indicando o caminho onde se encontra o ficheiro com o backup da base de dados. Depois, clique em “New” para criar a base de dados para onde os dados vão ser importados, atribuindo-lhe, novamente, o nome “futebol”, tal como mostra a imagem abaixo.



Por fim, clique no botão “Start Import” no final da página para iniciar a importação. A barra de progresso começará a avançar até estar totalmente preenchida de verde, que acontecerá quando o processo estiver concluído.

Se tudo correr bem, poderá voltar a utilizar a base de dados “futebol”. Para confirmá-lo, no menu do lado esquerdo, ao lado de “Schemas” está disponível o ícone de atualização da página. Ao clicar, a base de dados “futebol” deverá aparecer também na lista da base de dados.

Fazer cópias de segurança das bases de dados e restaurá-las não poderia ser mais fácil. De facto, deverá fazê-lo de vez em quando para evitar riscos desnecessários. Agora, poderá começar a trabalhar novamente com a base de dados “futebol”. E é exatamente isso que fará nos pontos seguintes.

2. VIEWS

Por vezes, no dia a dia, existem detalhes que se querem dar a conhecer e outros que não – o mundo do software não é exceção. Portanto, em certos casos, são definidas interfaces públicas, mantendo os detalhes da implementação privados. Nas bases de dados, é possível aplicar uma funcionalidade semelhante, de forma a garantir que os utilizadores não conhecem as tabelas das bases de dados, fornecendo-lhes apenas os detalhes que se pretende que saibam através das views, ou visualizações.

As views permitem consultar dados, mas, ao contrário das tabelas, não os armazenam. As views são criadas através de uma instrução **SELECT**, que é armazenada para uso posterior. Portanto, podem ser úteis quando pretende ocultar certas informações, mas também quando há consultas que são utilizadas com muita frequência.

Para saber como criar, modificar e eliminar views pode recorrer à ajuda do MySQL na linha de comandos ou visitar as seguintes páginas web com a documentação oficial:

- <http://dev.mysql.com/doc/refman/5.1/en/create-view.html>
- <http://dev.mysql.com/doc/refman/5.1/en/alter-view.html>
- <http://dev.mysql.com/doc/refman/5.1/en/drop-view.html>

```

Linha de comandos - mysql
mysql> help VIEW;
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
  ALTER VIEW
  CREATE VIEW
  DROP VIEW
mysql>

```

A sintaxe de criação de uma view é a seguinte:

CREATE

```

[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW nomeView [(listaColunas)]
AS consultaSeleccao
[WITH [CASCADED | LOCAL] CHECK OPTION]

```

Poderá colocá-lo em prática com uma das consultas utilizadas nas unidades anteriores: é a consulta que permitiu obter os campeões e segundos classificados dos diferentes títulos. Em seguida, apresenta-se a sintaxe de criação de uma view. A palavra reservada **REPLACE** é opcional, embora seja habitual utilizá-la, uma vez que, se houver outra com o mesmo nome, será substituída pela última.

```

CREATE OR REPLACE VIEW titulosView
AS SELECT
    CONCAT(CAST(temp.anoInicio AS CHAR(4)),
        ' - ',
        CAST(temp.anoFim AS CHAR(4)))
    AS Temporada,
    comp.competicaoNome AS Competição,
    eq1.nomeEquipa AS Campeão,
    eq2.nomeEquipa AS SegundoClassificado
FROM
    campeonato camp

```

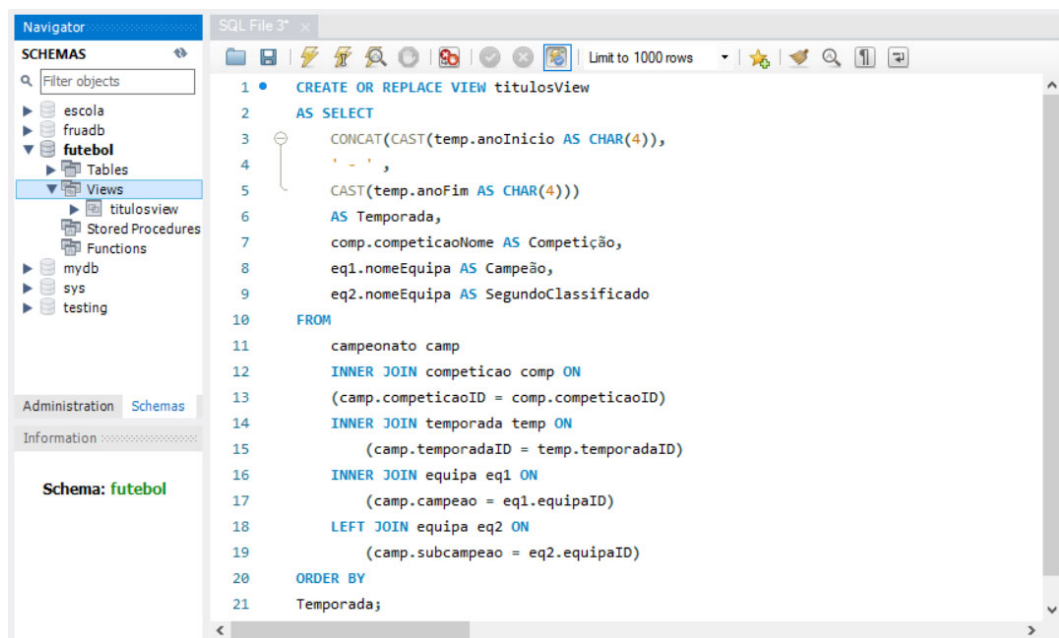
```

INNER JOIN competicao comp ON
(camp.competicaoID = comp.competicaoID)
INNER JOIN temporada temp ON
(camp.temporadaID = temp.temporadaID)
INNER JOIN equipa eq1 ON
(camp.campeao = eq1.equipaID)
LEFT JOIN equipa eq2 ON
(camp.subcampeao = eq2.equipaID)

ORDER BY
Temporada;

```

Se executar o código anterior no MySQL Workbench, será criada uma view que permitirá aceder aos mesmos dados a que iria aceder com essa mesma consulta. Contudo, como irá verificar, cada vez que pretender aceder-lhe, não será necessário escrever tanto código. Uma vez executado o código acima, clique novamente no ícone de atualização para atualizar a janela do Workbench e, desta forma, poderá ver a nova view que acabou de criar.



Uma vez que uma view tenha sido criada, poderá consultá-la com a instrução **SELECT** da mesma forma que foi feito com as tabelas nas unidades anteriores.

Neste caso, poderá testá-lo escrevendo a consulta apresentada abaixo. Verificará que o resultado é o mesmo de quando é executada a consulta sem a view, ou seja, a maior versão da consulta, utilizada nas unidades anteriores. Além de selecionar todos os registos, também é possível filtrar a pesquisa por determinados critérios.

```
SELECT * FROM titulosView;
```

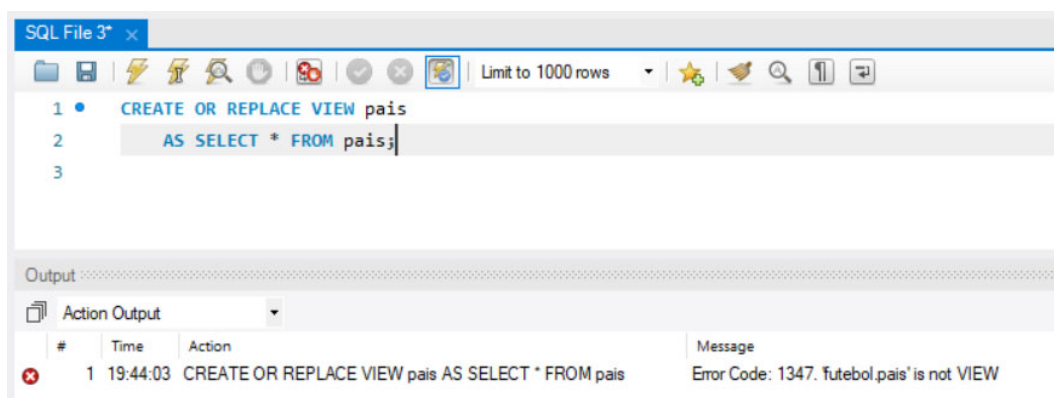
Se, ao criar a view, pretender nomear explicitamente as colunas da view, poderá fazê-lo colocando-as entre parênteses e separadas por vírgulas após o nome da view. Neste caso, como tal não foi efetuado, os nomes foram atribuídos diretamente na consulta.

É importante referir que não pode haver uma view com o mesmo nome de uma tabela, já que isso não permitiria saber exatamente o que se pretende consultar em todos os momentos, tal como pode confirmar ao tentar criar uma view da tabela “pais” com o mesmo nome.

```
CREATE OR REPLACE VIEW pais
AS SELECT * FROM pais;
```

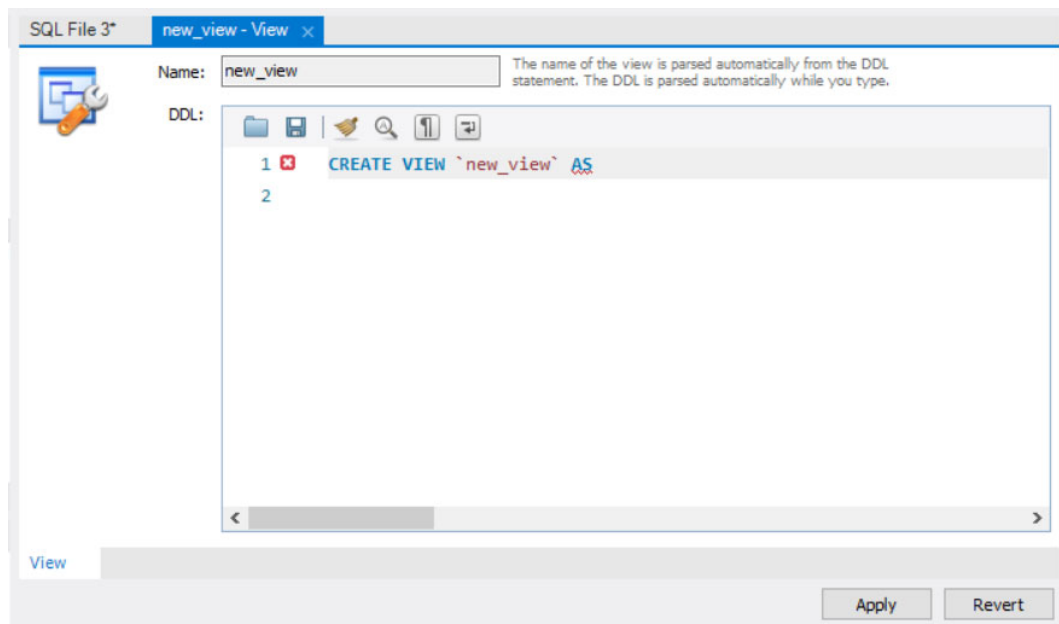
Ao executar o código acima, surge o seguinte erro:

```
Error Code: 1347
'futebol.pais' is not VIEW
```



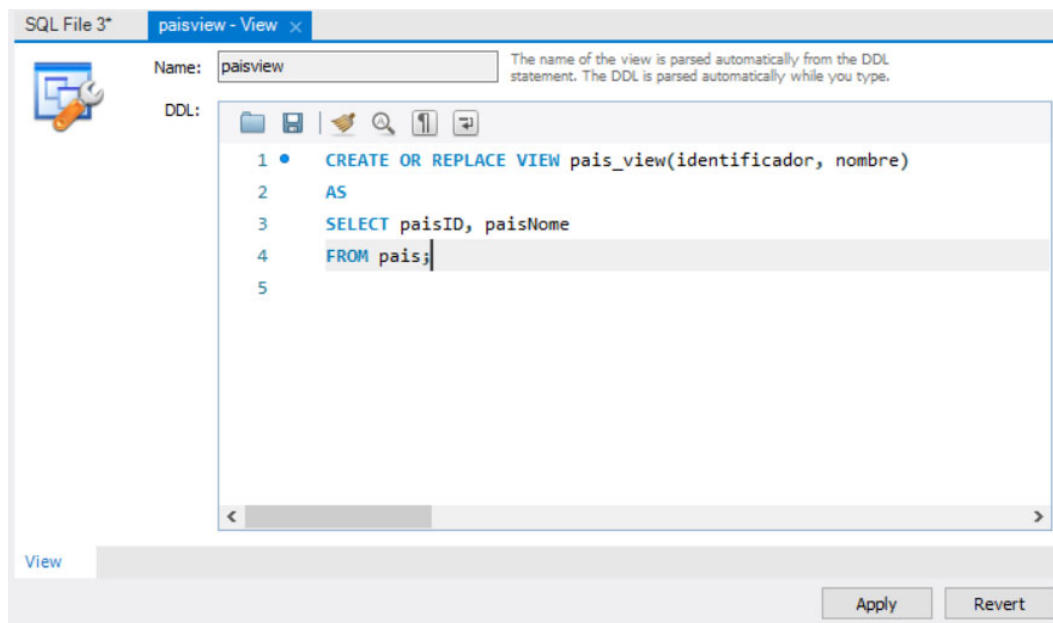
Também é possível criar views com uma pequena ajuda do MySQL Workbench, embora o modo gráfico não forneça muita ajuda desta vez. A melhor forma de o

fazer é clicando com o botão direito do rato na opção “Views” do menu à esquerda (“Schemas”) e escolher a opção “Create View”. Irá abrir-se uma janela com um modelo de texto com a sintaxe de criação de uma view, embora tenha de ser completada manualmente, e não automaticamente, como é habitual no modo gráfico.



Em seguida, tentará então criar-se a view “pais” novamente, mas, desta vez, com outro nome, para que não falhe nem dê erro. Além disso, serão alterados os nomes das colunas para que o utilizador não veja os nomes reais da tabela.

```
CREATE OR REPLACE VIEW paisView(identificador, nome)
AS
SELECT paisID, paisNome
FROM pais;
```



Embora no momento não seja preciso modificar nenhuma das views, é possível fazê-lo através da seguinte sintaxe de modificação de views (sintaxe muito semelhante à abordada em unidades anteriores).

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

A sintaxe de exclusão de uma view também é muito semelhante à visualizada em unidades anteriores:

```
DROP VIEW [IF EXISTS]
view_name [, view_name]
[RESTRICT | CASCADE]
```

3. SUBCONSULTAS OU SUBQUERIES

O MySQL oferece a possibilidade de executar consultas dentro de outras consultas; são as chamadas subqueries ou subconsultas. Este tipo de consulta está agrupado com a cláusula WHERE da consulta contida. Em seguida, verá melhor este caso, com alguns exemplos que mostrarão a utilidade e o poder deste tipo de consultas.

Por exemplo, para recuperar por ordem alfabética todos os nomes das equipas portuguesas registadas na base de dados, deve primeiro recuperar o identificador do país para Portugal e, em seguida, usá-lo como ponto de partida para a pesquisa na tabela "equipa". Poderá fazê-lo com a consulta apresentada abaixo, que irá devolver o valor do identificador do país.

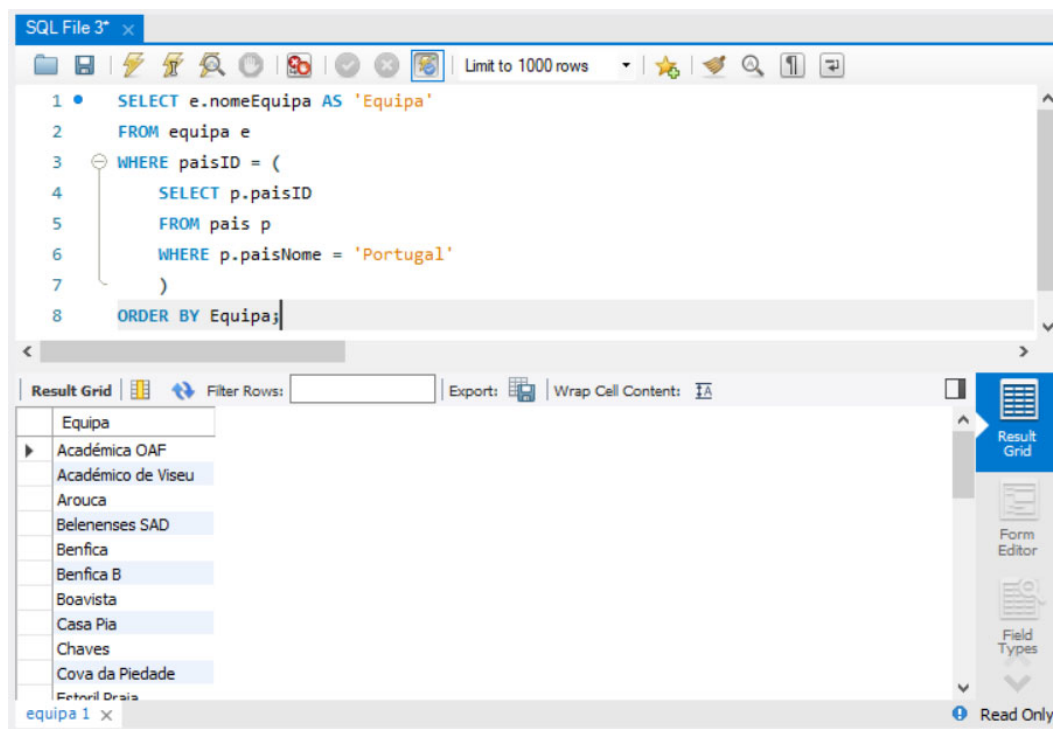
```
SELECT p.paisID
FROM pais p
WHERE p.paisNome = 'Portugal';
```

O resultado da consulta acima é "POR". Este valor poderia ser utilizado diretamente na consulta a seguir, que iria devolver todas as equipas portuguesas por ordem alfabética.

```
SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID = 'PT'
ORDER BY Equipa;
```

Resolver o problema desta maneira requer, portanto, duas etapas, duas consultas a serem executadas separadamente. No entanto, é possível resolver o problema de uma só vez utilizando subqueries, como mostra o exemplo a seguir. As duas consultas foram unidas para que funcionem juntas e a subquery estará delimitada entre parênteses.

```
SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID = (
    SELECT p.paisID
    FROM pais p
    WHERE p.paisNome = 'Portugal'
)
ORDER BY Equipa;
```

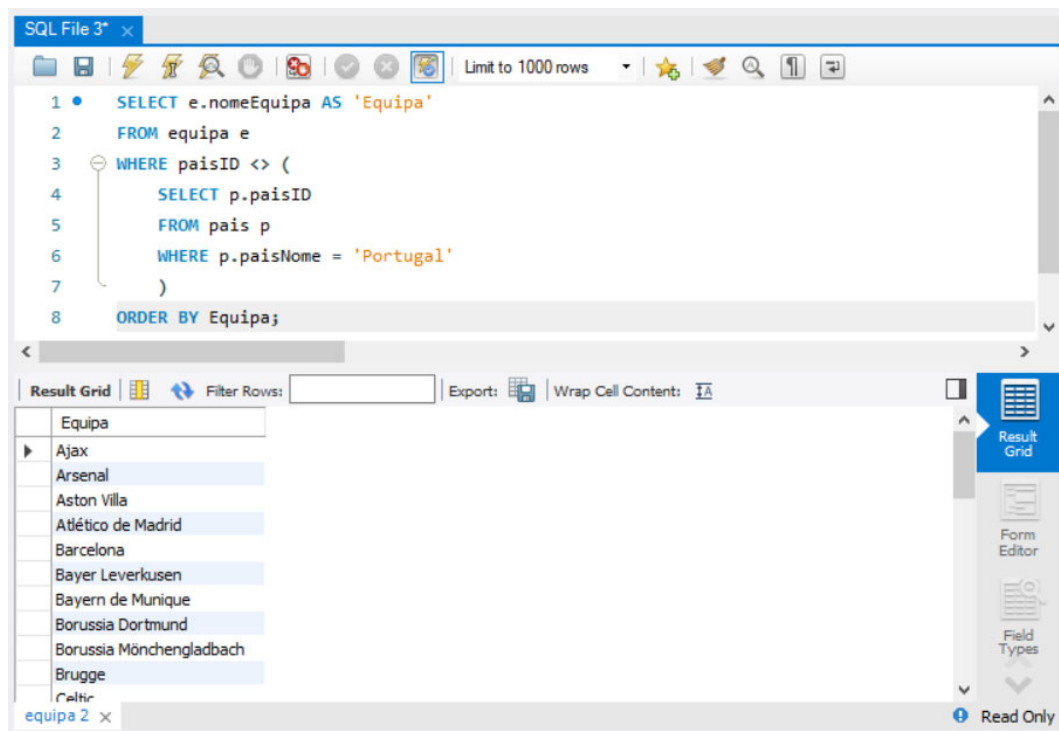


Como referido anteriormente, através do operador de igualdade ou atribuição (=) é possível obter os nomes das equipas portuguesas. Deste modo, para procurar e encontrar as equipas “não portuguesas” que se encontram registadas na base de dados, os operadores “<>” e “!=” deverão servir perfeitamente. Ou não?


```

SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID <> (
    SELECT p.paisID
    FROM pais p
    WHERE p.paisNome = 'Portugal'
)
ORDER BY Equipa;

```



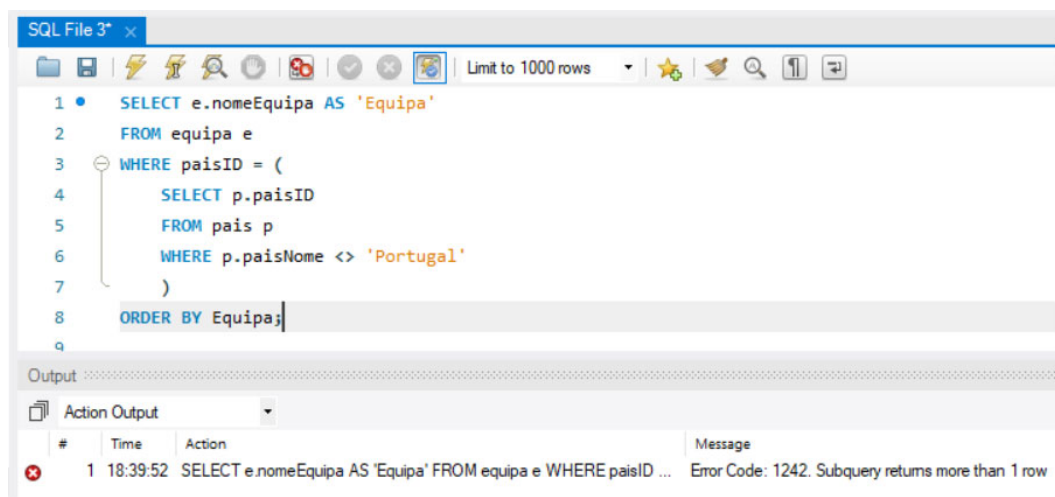
Como referido na última linha antes do exemplo, é possível obter as equipas estrangeiras registadas na nossa base de dados. Mas poderá estar a questionar-se se seria possível fazer o mesmo mantendo a consulta principal com o operador de igualdade e indicando na subconsulta que se pretende pesquisar os países que não têm Portugal como nome.

```
SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID = (
    SELECT p.paisID
    FROM pais p
    WHERE p.paisNome <> 'Portugal'
)
ORDER BY Equipa;
```

Embora esta abordagem pareça correta, verifica-se que, ao executar a consulta correspondente, surge o seguinte erro, indicando que a subconsulta devolve mais do que uma linha:

Error Code: 1242
Subquery returns more than 1 row

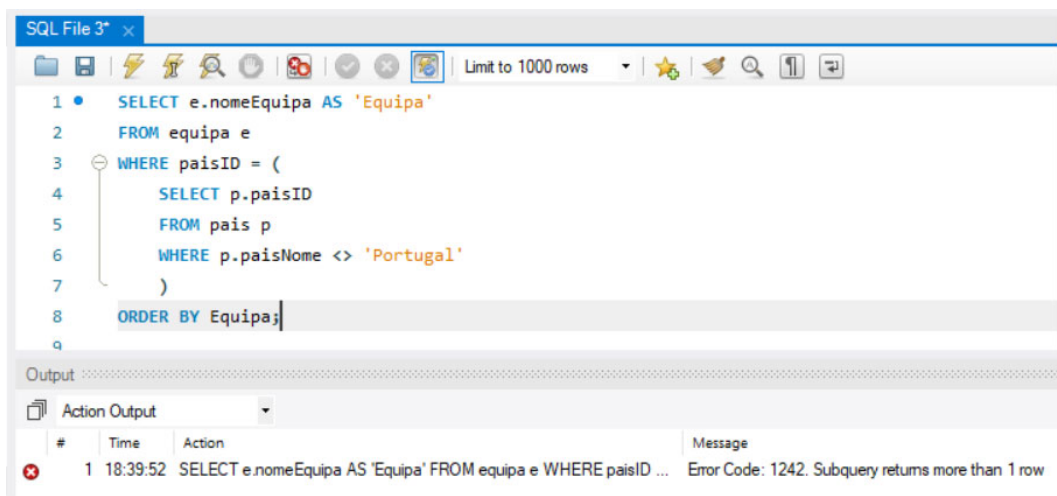
Na verdade, a subconsulta devolve mais do que uma linha, e é isso que se pretende, que devolva as linhas de todos os países, exceto Portugal. Mas acontece que a sintaxe utilizada não permite combinar um valor único ("paisID") com vários valores. A solução para este problema reside no operador **IN**, como é utilizado abaixo.



```
SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID IN (
    SELECT p.paisID
    FROM pais p
    WHERE p.paisNome <> 'Portugal'
)
ORDER BY Equipa;
```

Além da forma anterior de trabalhar com o operador IN, também é possível utilizá-lo de outras que permitirão trabalhar com uma lista de valores constantes. Por exemplo, se pretender encontrar as equipas de países que falem francês, como a França ou a Bélgica.

```
SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID IN ('FRA', 'BEL')
ORDER BY Equipa;
```



Da mesma forma que poderia ter interesse nos registos que contêm alguns dos valores fornecidos na lista, também poderia ter interesse nos valores que não estão na lista; nesse caso, poderá colocar a palavra **NOT** antes da palavra **IN**. Como sabe, a palavra inglesa “not” representa negação. Portanto, outra maneira de listar as equipas “não portuguesas” pode ser a seguinte:

```
SELECT e.nomeEquipa AS 'Equipa'
FROM equipa e
WHERE paisID NOT IN ('POR')
ORDER BY Equipa;
```

Para além do operador IN, existem outros operadores de interesse como: **ALL**, **SOME** e **ANY**. Estes permitem comparações com cada uma das linhas devolvidas. As palavras SOME e ANY são equivalentes e significam que a condição se torna verdadeira quando o valor de qualquer uma das linhas torna a comparação verdadeira, enquanto, no caso da palavra ALL, estas devem ser verdadeiras em todas as linhas. Pode encontrar mais informações sobre estes operadores em <http://dev.mysql.com/doc/refman/5.1/in/subqueries.html>.

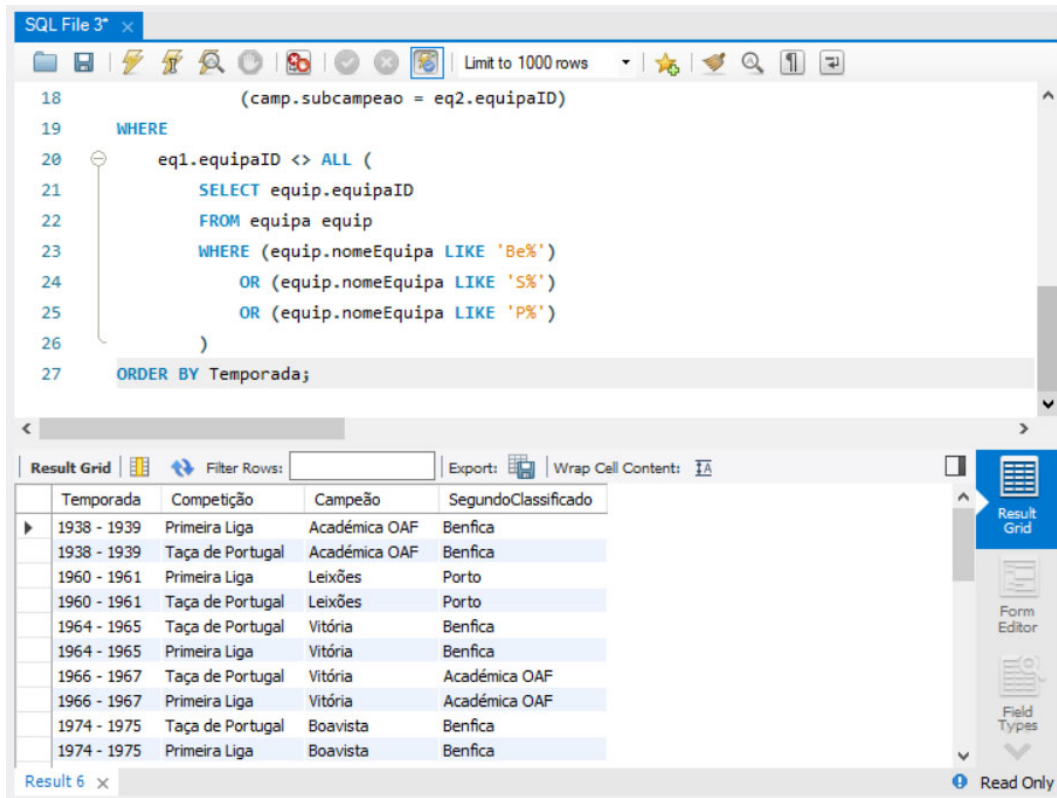
Segue-se um exemplo da utilização do operador ALL, que vai devolver todos os campeonatos que não foram vencidos por equipas cujo nome comece com "Be", "S" ou "P".

```
SELECT
    CONCAT(CAST(temp.anoInicio AS CHAR(4)),
           ' - ',
           CAST(temp.anoFim AS CHAR(4)))
    AS Temporada,
    comp.competicaoNome AS Competição,
    eq1.nomeEquipa AS Campeão,
    eq2.nomeEquipa AS SegundoClassificado
FROM
    campeonato camp
    INNER JOIN competicao comp ON
        (camp.competicaoID = comp.competicaoID)
    INNER JOIN temporada temp ON
        (camp.temporadaID = temp.temporadaID)
    INNER JOIN equipa eq1 ON
        (camp.campeao = eq1.equipaID)
    LEFT JOIN equipa eq2 ON
        (camp.subcampeao = eq2.equipaID)
```

```

WHERE
    eq1.equipaID <> ALL (
        SELECT equip.equipaID
        FROM equipa equip
        WHERE (equip.nomeEquipa LIKE 'Be%')
            OR (equip.nomeEquipa LIKE 'S%')
            OR (equip.nomeEquipa LIKE 'P%')
        )
ORDER BY Temporada;

```



The screenshot shows a MySQL IDE window with a SQL query editor and a results grid. The query is as follows:

```

18      (camp.subcampeao = eq2.equipaID)
19  WHERE
20      eq1.equipaID <> ALL (
21          SELECT equip.equipaID
22          FROM equipa equip
23          WHERE (equip.nomeEquipa LIKE 'Be%')
24              OR (equip.nomeEquipa LIKE 'S%')
25              OR (equip.nomeEquipa LIKE 'P%')
26          )
27  ORDER BY Temporada;

```

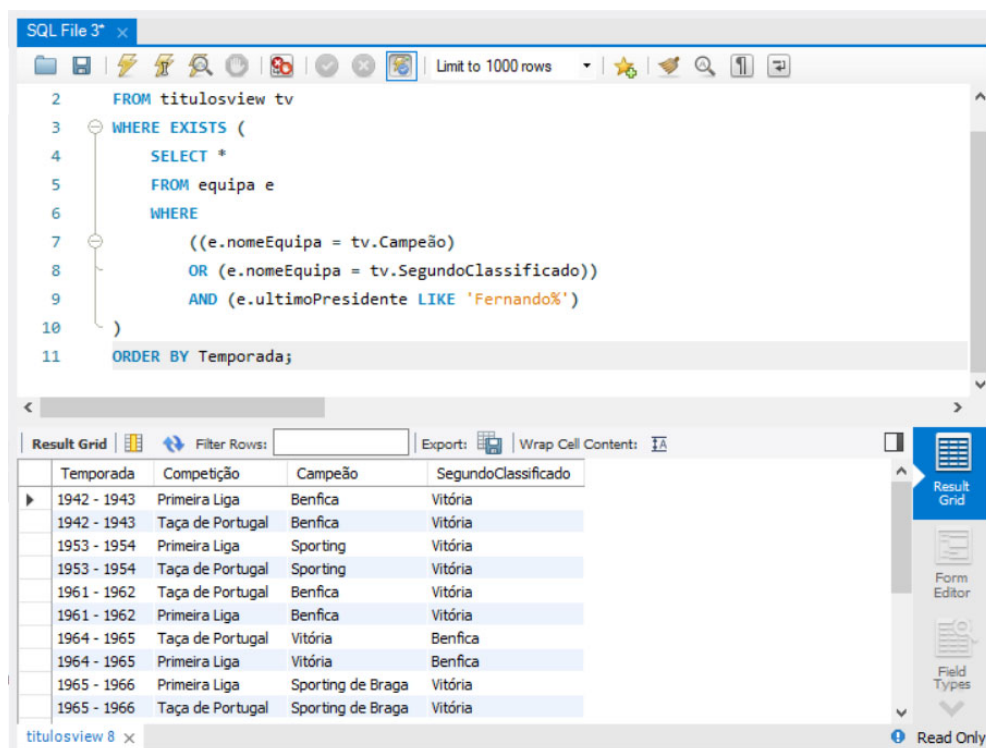
The results grid displays the following data:

Temporada	Competição	Campeão	SegundoClassificado
1938 - 1939	Primeira Liga	Académica OAF	Benfica
1938 - 1939	Taça de Portugal	Académica OAF	Benfica
1960 - 1961	Primeira Liga	Leixões	Porto
1960 - 1961	Taça de Portugal	Leixões	Porto
1964 - 1965	Taça de Portugal	Vitória	Benfica
1964 - 1965	Primeira Liga	Vitória	Benfica
1966 - 1967	Taça de Portugal	Vitória	Académica OAF
1966 - 1967	Primeira Liga	Vitória	Académica OAF
1974 - 1975	Taça de Portugal	Boavista	Benfica
1974 - 1975	Primeira Liga	Boavista	Benfica

Existem outras maneiras de agrupar consultas através dos operadores **EXISTS** e **NOT EXISTS**, que permitem verificar se o resultado da subquery está vazio. Mais informações sobre o seu uso podem ser encontradas em <http://dev.mysql.com/doc/refman/5.1/en/exists-and-not-exists-subqueries.html>.

Como exemplo, procurar-se-á por competições em que uma equipa tenha sido campeã ou segunda classificada e cujo último presidente se chame Fernando. Neste caso, o que se pretende é recuperar os dados das competições em que uma equipa cujo nome do seu último presidente é Fernando não apareça como campeã ou segunda classificada, logo, basta adicionar o NOT antes de EXISTS.

```
SELECT *
FROM titulosview tv
WHERE EXISTS (
    SELECT *
    FROM equipa e
    WHERE
        ((e.nomeEquipa = tv.Campeão)
        OR (e.nomeEquipa = tv.SegundoClassificado))
        AND (e.ultimoPresidente LIKE 'Fernando%')
)
ORDER BY Temporada;
```



The screenshot shows a SQL IDE window titled "SQL File 3" with a query editor and a result grid. The query is as follows:

```
2 FROM titulosview tv
3 WHERE EXISTS (
4     SELECT *
5     FROM equipa e
6     WHERE
7         ((e.nomeEquipa = tv.Campeão)
8         OR (e.nomeEquipa = tv.SegundoClassificado))
9         AND (e.ultimoPresidente LIKE 'Fernando%')
10 )
11 ORDER BY Temporada;
```

The result grid displays the following data:

Temporada	Competição	Campeão	SegundoClassificado
1942 - 1943	Primeira Liga	Benfica	Vitória
1942 - 1943	Taça de Portugal	Benfica	Vitória
1953 - 1954	Primeira Liga	Sporting	Vitória
1953 - 1954	Taça de Portugal	Sporting	Vitória
1961 - 1962	Taça de Portugal	Benfica	Vitória
1961 - 1962	Primeira Liga	Benfica	Vitória
1964 - 1965	Taça de Portugal	Vitória	Benfica
1964 - 1965	Primeira Liga	Vitória	Benfica
1965 - 1966	Primeira Liga	Sporting de Braga	Vitória
1965 - 1966	Taça de Portugal	Sporting de Braga	Vitória

The result grid is titled "Result Grid" and has a "Filter Rows" field. The query editor shows the query being executed, and the result grid shows the results of the query. The query is executed successfully, and the results are displayed in the result grid.

CONCLUSÃO

Nesta unidade didática, aprendeu a recuperar backups feitos anteriormente, assim como a criar e usar views, e realizar consultas agrupadas. Todos estes são elementos que os programadores e administradores de base de dados utilizam com regularidade, embora, sem dúvida, tenham muitos mais truques na manga do que os apresentados aqui. Portanto, aprenda tudo o que puder, tanto através de livros, quanto dos colegas da área que encontrará no seu percurso.

AUTOAVALIAÇÃO

1. Qual das seguintes afirmações é verdadeira?

- a) Os backups são inúteis.
- b) É aconselhável fazer cópias de segurança das bases de dados regularmente para protegê-las contra perda de informações.
- c) A recomendação de fazer backup das bases de dados é uma invenção das empresas de armazenamento físico de dados para ganharem mais dinheiro.
- d) É melhor não investir tempo ou dinheiro em cópias de segurança, pois os suportes informáticos onde estão armazenadas também podem ser danificados.

2. Indique a sintaxe correta para criar views:

- a) `CREATE VIEW nomeView AS (consulta);`.
- b) `GENERATE VIEW nomeView AS (consulta);`.
- c) `CREATE VIEW nomeView AS (nomeTabela);`.
- d) `GENERATE VIEW nomeView AS (nomeTabela);`.

3. **É possível criar uma view com o mesmo nome de uma tabela?**
 - a) Sim.
 - b) Sim, mas a view deve ser em maiúsculas.
 - c) Sim, mas a view deve ser escrita em letras minúsculas.
 - d) Não, porque partilham o mesmo namespace.

4. **Qual é a sintaxe correta para modificar uma view?**
 - a) MANIPULATE VIEW nomeView AS consulta;.
 - b) CHANGE VIEW nomeView AS consulta;.
 - c) ALTER VIEW nomeView AS consulta;.
 - d) MODIFY VIEW nomeView AS consulta;.

5. **Qual das seguintes afirmações permite excluir uma view?**
 - a) DROP VIEW nomeVista;.
 - b) DELETE VIEW nomeVista;.
 - c) ERASE VIEW nomeVista;.
 - d) REMOVE VIEW nomeVista;.

6. **Quais são os principais benefícios do uso das views?**
 - a) Permitem modificar os dados de uma tabela.
 - b) Oferecem vantagens consideráveis em termos de segurança da base de dados.
 - c) Permitem ocultar ao utilizador final determinados detalhes da base de dados, além de oferecerem uma interface simples para as consultas mais comuns.
 - d) Quase não têm vantagens sobre as tabelas, mas tornaram-se moda nos anos 1990.

7. Qual é a sintaxe adequada para utilizar uma view?
 - a) CALL nomeVista;.
 - b) SELECT * FROM nomeVista;.
 - c) USE nomeVista;.
 - d) PLAY nomeVista;.

8. Qual das seguintes palavras pode acompanhar a criação de uma view, para modificá-la se necessário?
 - a) RECALL.
 - b) REPLACE.
 - c) REPLAY.
 - d) REPLACE.

9. Em que cláusula podem ser adicionadas as subqueries?
 - a) SELECT.
 - b) FROM.
 - c) WHERE.
 - d) HAVING.

10. Qual é o delimitador utilizado para as subconsultas?
 - a) Plicas.
 - b) Parênteses.
 - c) Chavetas.
 - d) Aspas.

SOLUÇÕES

1.	b	2.	a	3.	d	4.	c	5.	a
6.	c	7.	b	8.	d	9.	c	10.	b

PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO

Para expandir e consolidar o conhecimento apreendido nesta unidade, visite as seguintes páginas web:

- https://www.w3schools.com/sql/sql_any_all.asp
- https://www.w3schools.com/sql/sql_exists.asp

BIBLIOGRAFIA

- Beaulieu, A. (2006), *Aprende SQL*. Madrid: Anaya Multimedia.
- Gutiérrez Gallardo, J. D. (2009), *MySQL 5.1*. Madrid: Anaya Multimedia.
- Oracle (2021), "Reference Manual". Disponível em: <http://dev.mysql.com/doc/refman/5.1/en/>.

Consultado a 5 de maio de 2021.

