

MÓDULO

BASE DE DADOS MYSQL

UNIDADE

MODIFICAÇÃO DE DADOS EM MYSQL

ÍNDICE

OBJETIVOS.....	3
INTRODUÇÃO.....	4
1. UPDATE	5
2. MODIFICAR AS TABELAS	15
CONCLUSÃO.....	23
AUTOAVALIAÇÃO.....	25
SOLUÇÕES.....	29
PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO.....	30
BIBLIOGRAFIA	31

OBJETIVOS

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

- Saber modificar a estrutura das tabelas que possuem os dados da base de dados.
- Aprender a modificar o tipo de dados utilizados numa determinada coluna de uma tabela, o seu comprimento máximo, se pode ou não aceitar valores nulos, entre outros aspetos.
- Saber atualizar os registos de uma tabela, corrigindo erros, completando-os com novas informações ou simplesmente aplicando as alterações que possam ter ocorrido.
- Aprender a adicionar novas colunas às tabelas da base de dados.

INTRODUÇÃO

Tal como o mundo muda com o tempo, as bases de dados também mudam. Tal acontece, porque o mundo que se conhece muda e estas são modelos dessa realidade.

Ao terminar de trabalhar nesta unidade, terá adquirido novos conhecimentos que lhe permitirão superar as pequenas dificuldades que encontra nas bases de dados utilizadas.

1. UPDATE

Todos os programas importantes incluem novas e constantes atualizações, seja por motivos de segurança, para adicionar novas funcionalidades aos programas, etc. Seja qual for o motivo, o mundo está em constante mudança e o mundo do software e das bases de dados não é exceção.

O MySQL oferece várias opções para realizar essas atualizações. Começando na linha de comandos, como abordado anteriormente, ao escrever **\help content;**, aparecerá, para consulta, a seguinte lista de categorias na janela:

- Account Management.
- Administration.
- Compound Statements.
- Data Definition.
- Data Manipulation.
- Data Types.
- Functions.
- Functions and Modifiers for Use with GROUP BY.
- Geographic Features.
- Help Metadata.
- Language Structure.
- Plugins.

- Table Maintenance.
- Transactions.
- User-Defined Functions.
- Utility.

Escolha a categoria **Data Manipulations**, porque é exatamente isso que se pretende trabalhar nesta unidade – manipular os dados e atualizá-los. A forma de obter a ajuda correspondente será escrever e executar **\help Data Manipulation**; O resultado será uma nova lista de categorias para escolher.

- CALL.
- DELETE.
- DO.
- DUAL.
- HANDLER.
- INSERT.
- INSERT DELAYED.
- INSERT SELECT.
- JOIN.
- LOAD DATA.
- REPLACE.
- SELECT.
- TRUNCATE TABLE.
- UNION.
- UPDATE.

De toda a lista de palavras em inglês, a que tem o significado de atualização é **UPDATE**. No entanto, algumas palavras da lista anterior já são conhecidas e já foram abordadas, como **INSERT**, **JOIN**, **SELECT** ou **UNION**. Será então executado **\help UPDATE**, de forma a obter a ajuda pretendida. Em <http://dev.mysql.com/doc/refman/5.1/en/update.html> poderá encontrar mais detalhes sobre a sintaxe desta função.


```
C:\> Linha de comandos - mysql -h 127.0.0.1 -P 8889 -u root -proot
mysql> \help UPDATE
Name: 'UPDATE'
Description:
Syntax:
UPDATE is a DML statement that modifies rows in a table.

Single-table syntax:

UPDATE [LOW_PRIORITY] [IGNORE] table_reference
      SET assignment_list
      [WHERE where_condition]
      [ORDER BY ...]
      [LIMIT row_count]

value:
      {expr | DEFAULT}

assignment:
      col_name = value

assignment_list:
      assignment [, assignment] ...

Multiple-table syntax:

UPDATE [LOW_PRIORITY] [IGNORE] table_references
      SET assignment_list
      [WHERE where_condition]

For the single-table syntax, the UPDATE statement updates columns of
existing rows in the named table with new values. The SET clause
indicates which columns to modify and the values they should be given.
Each value can be given as an expression, or the keyword DEFAULT to set
a column explicitly to its default value. The WHERE clause, if given,
specifies the conditions that identify which rows to update. With no
WHERE clause, all rows are updated. If the ORDER BY clause is
specified, the rows are updated in the order that is specified. The
LIMIT clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, UPDATE updates rows in each table named
in table_references that satisfy the conditions. Each matching row is
updated once, even if it matches the conditions multiple times. For
multiple-table syntax, ORDER BY and LIMIT cannot be used.
```

Como pode ver na imagem, a sintaxe possui duas cláusulas obrigatórias que são **UPDATE** e **SET**; as que estão entre chavetas são opcionais. Embora a cláusula **WHERE** seja opcional, recomenda-se que seja sempre utilizada, caso contrário, ocorrerá uma atualização massiva em todos os registos, o que na maior parte das vezes não é desejável.

As duas sintaxes utilizadas com mais frequência serão as seguintes: a primeira, quando se pretende apenas atualizar um registo específico ou uma série de registos que cumpram certas condições; a segunda, quando se pretende atualizar todos os registos de uma tabela. Embora ambas devam ser utilizadas com atenção, a segunda exige prudência redobrada.

```
UPDATE nomeTabela
SET
coluna1 = expressão1,
coluna2 = expressão2,
...,
colunaN = expressãoN
WHERE condições

UPDATE nomeTabela
SET
coluna1 = expressão1,
coluna2 = expressão2,
...,
colunaN = expressãoN
```

Em seguida, apresenta-se uma tabela com as datas em que as finais da Taça de Portugal foram disputadas entre as temporadas de 1995 a 2019. Além disso, são incluídos alguns dados que podem ser interessantes, como a hora de início da partida, o estádio onde foi disputada e a cidade onde o estádio está localizado.

Temporada	Data	Hora	Estádio
1995-1996	18/05/1996	20:30	Estádio Nacional
1996-1997	10/06/1997	21:00	Estádio Nacional
1997-1998	24/05/1998	21:00	Estádio Nacional
1998-1999	19/06/1999		Estádio Nacional
1999-2000	25/05/2000		Estádio Nacional
2000-2001	10/06/2001		Estádio Nacional
2001-2002	12/05/2002		Estádio Nacional

Temporada	Data	Hora	Estádio
2002-2003	15/06/2003		Estádio Nacional
2003-2004	16/05/2004	22:00	Estádio Nacional
2004-2005	29/05/2005	21:00	Estádio Nacional
2005-2006	14/05/2006	21:30	Estádio Nacional
2006-2007	27/05/2007	22:00	Estádio Nacional
2007-2008	18/05/2008	22:00	Estádio Nacional
2008-2009	31/05/2009	22:00	Estádio Nacional
2009-2010	16/05/2010	21:30	Estádio Nacional
2010-2011	22/05/2011		Estádio Nacional
2011-2012	20/05/2012		Estádio Nacional
2012-2013	26/05/2013		Estádio Nacional
2013-2014	18/05/2014		Estádio Nacional
2014-2015	31/05/2015		Estádio Nacional
2015-2016	22/05/2016		Estádio Nacional
2016-2017	28/05/2017		Estádio Nacional
2017-2018	20/05/2018		Estádio Nacional
2018-2019	25/05/2019		Estádio Nacional
2019-2020	01/08/2020		Estádio Cidade de Coimbra

Dos dados anteriores, o mais relevante é a data, que é um dado que se pode facilmente atualizar nos registos da tabela “campeonato”. Pode atualizar os registos facilmente se conhecer o “campeonatoID”, uma vez que se trata do identificador único. Na consulta para obter este identificador, bastaria alterar o ano para obter os diferentes identificadores. As temporadas em que foram atribuídos dois títulos são a exceção, e para os distinguir teria de adicionar mais algum filtro, como, por exemplo, a equipa vencedora.

```
SELECT camp.campeonatoID
FROM
campeonato camp
INNER JOIN competicao comp ON
(camp.competicaoID = comp.competicaoID)
```

```
INNER JOIN temporada temp ON  
(camp.temporadaID = temp.temporadaID)  
WHERE  
(comp.competicaoNome LIKE 'Taça%')  
AND (temp.anoInicio=1995);
```

Uma vez que o identificador do campeonato é conhecido, deve ser inserido na consulta de atualização. Embora não seja necessário, foi utilizada uma abreviação (c) na consulta para que possa ser usado sem problemas.

```
UPDATE campeonato c  
SET c.data = '1996-05-18'  
WHERE c.campeonatoID = 136;
```

Repita esta operação tantas vezes quantos os resultados a serem atualizados. Porque esta operação é mais viável através do Workbench, deixá-la-emos para mais tarde. Por enquanto, basta que veja se funciona e o verifique após a atualização, como mostra a figura a seguir.

```

mysql> use futebol
Database changed
mysql> SELECT camp.campeonatoID
  -> FROM
  -> campeonato camp
  -> INNER JOIN competicao comp ON
  -> (camp.competicaoID = comp.competicaoID)
  -> INNER JOIN temporada temp ON
  -> (camp.temporadaID = temp.temporadaID)
  -> WHERE
  -> (comp.competicaoNome LIKE 'Taça%')
  -> AND (temp.anoInicio=1995);
+-----+
| campeonatoID |
+-----+
|          136 |
+-----+
1 row in set (0.00 sec)

mysql> UPDATE campeonato c
  -> SET c.data = '1996-05-18'
  -> WHERE c.campeonatoID = 136;
Query OK, 1 row affected (0.81 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT
  -> camp.campeonatoID,
  -> temp.anoInicio,
  -> temp.anoFim,
  -> camp.data
  -> FROM
  -> campeonato camp
  -> INNER JOIN competicao comp ON
  -> (camp.competicaoID = comp.competicaoID)
  -> INNER JOIN temporada temp ON
  -> (camp.temporadaID = temp.temporadaID)
  -> WHERE
  -> (comp.competicaoNome LIKE 'Taça de Portugal')
  -> AND (temp.anoInicio=1995);
+-----+-----+-----+-----+
| campeonatoID | anoInicio | anoFim | data      |
+-----+-----+-----+-----+
|          136 |       1995 |    1996 | 1996-05-18 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

Antes, foi atualizado apenas um registo, no entanto é possível atualizar vários registos com apenas uma consulta. Quanto maior o número de registos, maior o risco de alterar dados por engano, por isso é sempre recomendado muito cuidado ao fazer este tipo de consultas.

```
UPDATE temporada  
SET anoInicio = anoInicio + 1;
```

Ao executá-la, será exibida uma mensagem de erro indicando que não é possível fazê-lo devido aos valores duplicados. Tal acontece porque o primeiro registo tenta aumentar o seu valor numa unidade, mas, como o registo seguinte já tem esse valor, está a violar-se a restrição de integridade (não pode haver temporadas repetidas). Em seguida, faremos uma nova tentativa com a coluna "anoFim", subtraindo uma unidade de cada registo.

```
UPDATE temporada  
SET anoFim = anoFim - 1;
```

Desta vez, não haverá problemas porque, ao tentar subtrair uma unidade do primeiro registo, não há nenhum outro registo que tenha esse valor e, ao passar para o seguinte, o primeiro perdeu o seu valor antigo, logo, não há valores duplicados em nenhum momento. Desta forma, é possível alterar todos os valores da tabela. Embora não tenha havido nenhum problema em executar a consulta desta forma, seria recomendável fazê-lo com a cláusula **ORDER BY** para garantir que as atualizações são feitas do valor menor para o maior.

A maior dificuldade vem depois, quando, ao tentar colocar os registos como estavam antes, o problema de valores duplicados surge novamente. Uma das soluções seria remover a restrição de integridade da tabela, realizar as atualizações e atualizar a tabela novamente, mas isso seria muito arriscado porque outra pessoa poderia arruinar tudo. Então, a solução consiste em começar a fazer as atualizações não pelo primeiro registo, mas sim pelo último. Desta forma, o último registo perde o seu antigo valor, deixando-o disponível para o penúltimo, e assim sucessivamente.

```
UPDATE temporada
SET anoFim = anoFim + 1
ORDER BY anoFimDESC;
```

```
mysql> UPDATE temporada
  -> SET anoInicio = anoInicio + 1;
ERROR 1062 (23000): Duplicate entry '1902' for key 'anoInicio_UNIQUE'
mysql> UPDATE temporada
  -> SET anoFim = anoFim - 1;
Query OK, 121 rows affected (0.42 sec)
Rows matched: 121  Changed: 121  Warnings: 0

mysql> SELECT * FROM temporada LIMIT 5;
+-----+-----+-----+
| temporadaID | anoInicio | anoFim |
+-----+-----+-----+
|          1 |        1901 |        1901 |
|          2 |        1902 |        1902 |
|          3 |        1903 |        1903 |
|          4 |        1904 |        1904 |
|          5 |        1905 |        1905 |
+-----+-----+-----+
5 rows in set (0.02 sec)

mysql> UPDATE temporada
  -> SET anoFim = anoFim + 1
  -> ORDER BY anoFim DESC;
Query OK, 121 rows affected (0.19 sec)
Rows matched: 121  Changed: 121  Warnings: 0

mysql> SELECT * FROM temporada LIMIT 5;
+-----+-----+-----+
| temporadaID | anoInicio | anoFim |
+-----+-----+-----+
|          1 |        1901 |        1902 |
|          2 |        1902 |        1903 |
|          3 |        1903 |        1904 |
|          4 |        1904 |        1905 |
|          5 |        1905 |        1906 |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

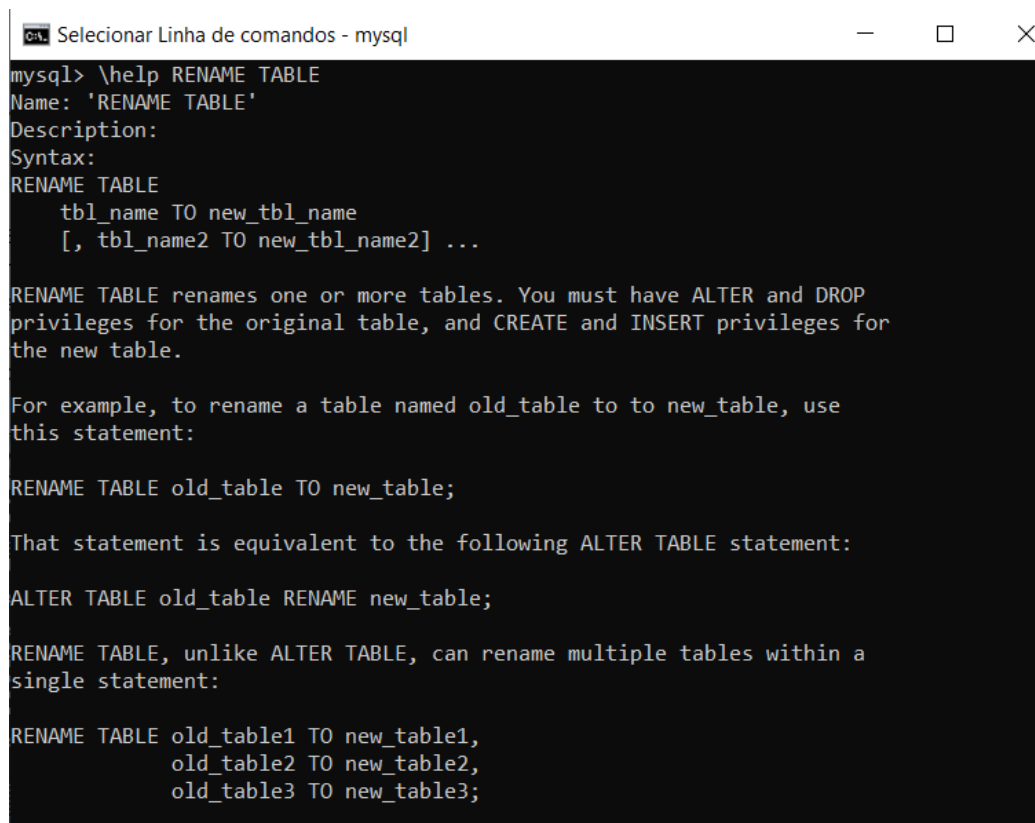
Como pôde observar, as consultas de atualização oferecem muitas possibilidades, mas devem ser executadas com cuidado, de modo a manter as bases de dados em boas condições.

Neste caso, as restrições de integridade funcionaram bem, e isso é algo positivo, uma vez que protegeram a base de dados de potenciais contratempos.

2. MODIFICAR AS TABELAS

Até agora, foram feitas alterações nos dados das tabelas. Neste ponto, aprenderá a fazer alterações nas próprias tabelas e na sua estrutura. Essas mudanças não são incomuns, já que, por vezes, surge a necessidade de adicionar novas colunas a uma tabela para registrar novos dados ou adicionar novas restrições de integridade que foram ignoradas numa primeira abordagem.

Embora seja incomum mudar o nome de uma tabela, uma vez que esta já está a ser usada há algum tempo, o MySQL oferece essa possibilidade através do comando **RENAME TABLE**. Mais informações sobre este comando podem ser encontradas em <http://dev.mysql.com/doc/refman/5.0/en/rename-table.html> e na ajuda da linha de comandos do MySQL.



```
Selecionar Linha de comandos - mysql
mysql> \help RENAME TABLE
Name: 'RENAME TABLE'
Description:
Syntax:
RENAME TABLE
    tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...

RENAME TABLE renames one or more tables. You must have ALTER and DROP
privileges for the original table, and CREATE and INSERT privileges for
the new table.

For example, to rename a table named old_table to to new_table, use
this statement:

RENAME TABLE old_table TO new_table;

That statement is equivalent to the following ALTER TABLE statement:

ALTER TABLE old_table RENAME new_table;

RENAME TABLE, unlike ALTER TABLE, can rename multiple tables within a
single statement:

RENAME TABLE old_table1 TO new_table1,
               old_table2 TO new_table2,
               old_table3 TO new_table3;
```

No entanto, a ordem para fazer alterações numa tabela é, por excelência, primeiro fazer **ALTER TABLE**, para a qual, como habitualmente, é obtida a ajuda correspondente através de `\help ALTER TABLE`;

Depois de executar esta frase, aparecerá a ajuda apresentada abaixo, que, como é possível ver, se trata de uma ordem muito versátil. Entre todas as cláusulas que **ALTER TABLE** contém, existe uma que também permite a renomeação da tabela, designada **RENAME TO**.

```
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
alter_specification [, alter_specification]
alter_specification:
table_options
| ADD [COLUMN] col_name column_definition
[FIRST | AFTER col_name ]
| ADD [COLUMN] (col_name column_definition,...)
| ADD {INDEX|KEY} [index_name]
```

```

[index_type] (index_col_name,...) [index_option] ...
| ADD [CONSTRAINT [symbol]] PRIMARY KEY
[index_type] (index_col_name,...) [index_option] ...
| ADD [CONSTRAINT [symbol]]
UNIQUE [INDEX|KEY] [index_name]
[index_type] (index_col_name,...) [index_option] ...
| ADD FULLTEXT [INDEX|KEY] [index_name]
(index_col_name,...) [index_option] ...
| ADD SPATIAL [INDEX|KEY] [index_name]
(index_col_name,...) [index_option] ...
| ADD [CONSTRAINT [symbol]]
FOREIGN KEY [index_name] (index_col_name,...)
reference_definition
| ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
| CHANGE [COLUMN] old_col_name new_col_name column_definition
[FIRST|AFTER col_name]
| MODIFY [COLUMN] col_name column_definition
[FIRST | AFTER col_name]
| DROP [COLUMN] col_name
| DROP PRIMARY KEY
| DROP {INDEX|KEY} index_name
| DROP FOREIGN KEY fk_symbol
| DISABLE KEYS
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name [, col_name] ...
| CONVERT TO CHARACTER SET charset_name
[COLLATE collation_name]
| [DEFAULT] CHARACTER SET [=] charset_name
[COLLATE [=] collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE

```

```
| partition_options
| ADD PARTITION (partition_definition)
| DROP PARTITION partition_names
| COALESCE PARTITION number
| REORGANIZE PARTITION [partition_names INTO
(partition_definitions)]
| ANALYZE PARTITION {partition_names | ALL }
| CHECK PARTITION {partition_names | ALL }
| OPTIMIZE PARTITION {partition_names | ALL }
| REBUILD PARTITION {partition_names | ALL }
| REPAIR PARTITION {partition_names | ALL }
| PARTITION BY partitioning_expression
| REMOVE PARTITIONING

index_col_name:
col_name [(length)] [ASC | DESC]

index_type:
USING {BTREE | HASH}

index_option:
KEY_BLOCK_SIZE [=] value
| index_type
| WITH PARSER parser_name

table_options:
table_option [[,] table_option] ... (see CREATE TABLE options)
```

Entre todas as possibilidades que o comando **ALTER TABLE** oferece, as mais utilizadas são aquelas que permitem adicionar elementos (**ADD**) e excluí-los (**DROP**). Em seguida, serão abordadas as principais opções que são, geralmente, usadas com ALTER TABLE no dia a dia.

Se pretender adicionar uma coluna a uma tabela, poderá fazê-lo com as indicações anteriores. No entanto, como exemplo, será adicionada uma nova coluna à tabela “equipa”, com a possibilidade de incluir informação sobre a data de fundação das equipas.

```
ALTER TABLE nomeTabela
ADD COLUMN nomeColuna tipoColuna(tamanho);

ALTER TABLE equipa
ADD COLUMN dataFundacao DATE;
```

Ao executar o comando anterior, uma coluna chamada “dataFundacao” do tipo DATE é adicionada à tabela “equipa”. Como a posição na qual inserir a coluna não foi especificada, esta é colocada no final. O MySQL possui as palavras reservadas **FIRST** e **AFTER** para indicar onde colocar a coluna, embora este detalhe possa não ser relevante.

No entanto, é possível observar que seria melhor usar o tipo **DATETIME** para a coluna recém-adicionada, porque, além da data, também se pretende registar a hora exata. Para isso, pode utilizar-se **MODIFY COLUMN**, como exemplificado abaixo.

```
ALTER TABLE equipa
MODIFY COLUMN datafundacao DATETIME;
```

Se, além de alterar a definição da coluna, quiser alterar também o seu nome, poderá fazê-lo com a opção **CHANGE COLUMN**. Por exemplo, se pretender que a data seja do tipo DATE novamente e que o nome da coluna seja “dataFund”, poderá fazê-lo executando o seguinte comando:

```
ALTER TABLE equipa
CHANGE COLUMN dataFundacao dataFund DATE;
```

Outra das operações comuns nas colunas pode ser a eliminação. Neste caso, será utilizada a opção **DROP COLUMN**. Para continuar com o mesmo exemplo, irá eliminar-se a coluna “dataFund” da tabela “equipa”. E logo a seguir esta será novamente adicionada, uma vez que se trata de uma coluna relevante para a base de dados.

```
ALTER TABLE equipa  
DROP COLUMN dataFund;  
  
ALTER TABLE equipa  
ADD COLUMN dataFundacao DATE;
```

Muitas vezes, além de modificar, adicionar ou remover colunas, é necessário durante a fase de desenvolvimento do software fazer alterações nas chaves primárias, chaves estrangeiras e índices de tabelas. Segue-se uma lista das principais variantes de ALTER TABLE que permitem este tipo de modificações, embora não vá usar nenhum dos exemplos nesta base de dados.

- **ADD INDEX:** adiciona um índice à tabela na coluna ou colunas especificadas.
- **ADD CONSTRAINT PRIMARY KEY:** converte a coluna indicada na chave primária da tabela.
- **ADD CONSTRAINT UNIQUE INDEX:** converte a coluna especificada numa coluna que não permite valores repetidos.
- **ADD CONSTRAINT FOREIGN KEY:** converte a coluna especificada numa chave secundária que aponta para a chave primária da tabela especificada.
- **DROP PRIMARY KEY:** remove a chave primária da tabela, mas não a coluna.
- **DROP INDEX:** remove o índice indicado, mas não a coluna correspondente.
- **DROP FOREIGN KEY:** exclui a chave externa ou secundária, mas não a coluna correspondente.
- **RENAME TO:** permite alterar o nome da tabela, embora não seja uma operação comum.

Como pode ser visto na sintaxe fornecida pela ajuda, existem muitas variantes do comando ALTER TABLE, mas raramente é necessário usá-las; por isso, é preferível aprender as mais úteis e procurar informações sobre as restantes em casos específicos e hipotéticos em que possam ser úteis. As que foram apresentadas são mais do que suficientes para o dia a dia.

CONCLUSÃO

Nesta unidade didática trabalhou na linha de comandos com o intuito de modificar as informações contidas nas tabelas da base de dados dedicadas ao futebol, bem como na própria estrutura das tabelas. Desta forma, é possível adaptar a base de dados às necessidades de cada momento, fazendo as extensões que se julguem adequadas sempre que possível.

Ao longo do curso foi criada uma base de dados com as suas tabelas e as suas relações, inseridos os registos correspondentes às informações disponíveis, recuperadas as informações e, finalmente, nesta unidade foram executadas as alterações pretendidas.

AUTOAVALIAÇÃO

1. **Qual das seguintes afirmações permite modificar os dados contidos numa coluna de uma tabela?**
 - a) ALTER nomeTabela SET coluna1=valor;.
 - b) UPDATE coluna1 FROM nomeTabela SET coluna1=valor;.
 - c) ALTER dado1 FROM nomeTabela SET coluna1=valor;.
 - d) UPDATE nomeTabela SET coluna1=valor;.

2. **Com qual das seguintes cláusulas de UPDATE é possível indicar que apenas os registos que cumprem certas condições são atualizados?**
 - a) LIMIT.
 - b) MAX.
 - c) WHERE.
 - d) IGNORE.

3. **O que é que a cláusula ORDER BY permite fazer num UPDATE?**
 - a) Nada, pois na versão utilizada do MySQL ainda não existe essa funcionalidade.
 - b) Permite que os registos sejam atualizados na ordem especificada.
 - c) Permite limitar o número de registos a atualizar.
 - d) Permite adicionar condições à consulta de atualização.

- 4. O que é que a cláusula LIMIT permite fazer num UPDATE?**
- a) Nada, pois na versão utilizada do MySQL ainda não existe essa funcionalidade.
 - b) Permite que os registos sejam atualizados na ordem especificada.
 - c) Permite limitar o número de registos a atualizar.
 - d) Permite adicionar condições à consulta de atualização.
- 5. Qual das seguintes afirmações permite fazer alterações na estrutura de uma tabela?**
- a) ALTER TABLE nomeTabela.
 - b) MODIFY nomeTabela.
 - c) RENAME TABLE nomeTabela.
 - d) UPDATE nomeTabela.
- 6. Qual das seguintes afirmações permite alterar o nome de uma tabela?**
- a) ALTER TABLE nomeTabela TO nomeNovoTabela.
 - b) MODIFY nomeTabela TO nomeNovoTabela.
 - c) RENAME TABLE nomeTabela TO nomeNovoTabela.
 - d) UPDATE nomeTabela TO nomeNovoTabela.
- 7. Qual é a sintaxe mais apropriada se se pretender adicionar uma coluna do tipo CHAR(10) a uma tabela?**
- a) ALTER TABLE nomeTabela ADD coluna1 CHAR(10).
 - b) INSERT INTO nomeTabela COLUMN coluna1.
 - c) UPDATE TABLE nomeTabela ADD coluna1 CHAR(10).
 - d) MODIFY nomeTabela ADD coluna1 CHAR(10).

8. Qual é a palavra utilizada com o ALTER TABLE para remover uma coluna de uma tabela?
- a) ERASE.
 - b) DELETE.
 - c) REMOVE.
 - d) DROP.
9. Com qual das seguintes afirmações é possível alterar o nome de uma tabela?
- a) ALTER TABLE nomeTabela RENAME TO nomeNovoTabela.
 - b) MODIFY nomeTabela TO nomeNovoTabela.
 - c) UPDATE nomeTabela TO nomeNovoTabela.
 - d) CHANGE TABLENAME FROM nomeTabela to nomeNovoTabela.
10. O que é alterado se a instrução ALTER TABLE for usada com RENAME TO?
- a) Nada, porque essa combinação não existe.
 - b) O nome da coluna.
 - c) O nome da base de dados.
 - d) O nome da tabela.

SOLUÇÕES

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

PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO

Para ampliar os conhecimentos aprendidos nesta unidade, recomenda-se a visita às seguintes páginas web:

- https://www.w3schools.com/sql/sql_alter.asp
- https://www.w3schools.com/sql/sql_drop_table.asp
- https://www.w3schools.com/sql/sql_update.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 27 de abril de 2021.