code4sp
coding for social promotion

Co-funded by the
Erasmus+ Programme
of the European Union

1.:

Pacote de Material de Formação Code4SP

WP3:

Materiais de Formação Code4SP

Preparado por:

CITIZENS IN POWER

Informação do Projeto

Acrónimo do projeto: Code4SP

Título do projeto: Coding for Social Promotion

Referência do projeto: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN

Website do projeto: www.code4sp.eu

Parceiro autor: CIP

Versão do documento: 1

Data de preparação: 25/02/2022

Histórico do Documento			
Data	Versão	Autor	Descrição
25/02/2022	1	CIP	Esboço

Tabela de Conteúdos

Informação do Projeto 2
Informação sobre o tema 6
4. SQL 6
Pré-requisitos: 6
Duração: 6
Descrição: 6
Objetivos de aprendizagem: 6
Material necessário: 6
Cenário de aula: 7
Subtemas: 7
Recursos adicionais: 7
4.1. Noções básicas de SQL 8
O que é a SQL? 8
Usos da SQL 8
A síntaxe da SQL 8
O uso de maiúsculas e minúsculas em SQL 9
O comando 'Select' em SQL 10
O comando 'SELECT DISTINCT' em SQL 11
O comando 'WHERE' em SQL 13
Os operadores 'AND', 'OR' e 'NOT' em SQL 14
O comando 'ORDER BY' em SQL 17
O comando 'INSERT INTO' em SQL 18
Valores 'NULL' em SQL 19
O comando 'UPDATE' em SQL 21
O comando 'DELETE' em SQL 22
O comando 'SELECT TOP' em SQL 23
A função 'MIN' e 'MAX' em SQL 26
As funções 'COUNT', 'AVG' E 'SUM' EM SQL 27
O operador 'LIKE' em SQL 29

Wildcards em SQL 31
O operador 'IN' em SQL 34
O operador 'BETWEEN' em SQL 36
Aliases em SQL 39
A operação 'JOIN' em SQL 40
'INNER JOIN' em SQL 42
'LEFT JOIN' em SQL 43
'RIGHT JOIN' em SQL 44
'FULL JOIN' em SQL 45
'SELFJOIN' em SQL 47
'UNION' em SQL 48
'GROUP BY' em SQL 51
'HAVING' em SQL 53
'SELECT INTO' em SQL 55
'INSERT INTO SELCT' em SQL 57
'CASE' em SQL 58
Funções 'NULL' em SQL 60
Comentários em SQL 62
Operadores SQL 63
4.2. Bases de Dados SQL 66
'CREATE DATABASE' em SQL 66
'DROP DB' em SQL 66
'BACKUP DB' em SQL 67
'CREATE TABLE' em SQL 68
'DROP TABEL' em SQL 70
'ALTER TABLE' em SQL 70
'CONSTRAINTS' em SQL 73
'NOT NULL' em SQL 74
A limitação 'UNIQUE' em SQL 75
A limitação 'PRIMARY KEY' em SQL 76
A limitação 'FOREIGN KEY' em SQL 78
A limitação 'CHECK' em SQL 80
A limitação 'DEFAULT' em SQL 82
O comando 'CREATE INDEX' em SQL 84

Auto Increment em SQL 86
Datas em SQL 89
Views em SQL 91
4.3. Referências SQL 93
Palavras-chave SQL 93
Funções em MySQL 101
Funções em SQL Server 101
Funções em MS Access 101
SQL Quick Ref 101
dados em SQL 102
4.4. Exemplos SQL 108
Exemplos SQL 108
Questionário SQL 108

Informação sobre o tema

Tema:

4. SQL

Pré-requisitos:

Literacia informática básica, *software* básico instalado e conhecimento básico sobre como trabalhar com ficheiros.

Duração:

10 horas

Descrição:

Este tema cobre as noções básicas de SQL, de forma a familiarizar os aprendentes com o mundo da programação e encorajá-los a ganhar competências de SQL. São explicados os atributos, síntaxe e outros termos relevantes que os aprendentes já poderão ter ouvido ou com que poderão estar familiarizados, e ainda de que forma estes se enquadram na linguagem de programação. É ainda facultada uma detalhada descrição da lógica e síntaxe usadas em SQL, a sua estrutura e outras funções básicas e essenciais.

Objetivos de aprendizagem:

- Reconhecer o conceito e uso da SQL em Sistemas de Gestão de Bases de Dados Relacionais (SGBDR);
- Formular síntaxe básica para consulta SQL;
- Usar a SQL para aceder, gerir e manipular SGBD.

Material necessário:

- Computador ou portátil;
- Ligação à internet;

- Compilador *online* SQL (https://www.mycompiler.io/new/sql);
- Editor de texto *online* (https://www.w3schools.com/sql/default.asp);

Cenário de aula:

No total, serão dedicadas 10 horas a este tema, sendo que fica ao critério de cada formador(a)/professor(a) decidir o tempo a alocar a cada subtema. Sugerimos o uso das apresentações PowerPoint sobre este tema criadas para a formação para facilitar o processo de ensino e aumentar a eficiência temporal. Estas apresentações abrangem o seguinte:

- Desenvolvimento progressivo dos subtemas e conceitos centrais a reter;
- Exercícios recomendados.

Consoante as preferências do(a) formador(a)/professor(a), o desenvolvimento progressivo das apresentações permite a conclusão da sessão SQL no tempo estipulado, 10 horas. As apresentações também podem ser disponibilizadas aos formandos para estudo individual.

Subtemas:

- 4.1. Noções básicas de SQL
- 4.2. Bases de Dados SQL
- 4.3. Referências SQL
- 4.4. Exemplos de SQL

Recursos adicionais:

- W3Schools – Guia sobre todas as palavras-chave e funções de SQL e exemplos de cada um deles;
- Tutorialspoint – Outro guia detalhado sobre palavras-chave e funções SQL e vários exemplos de cada um deles.

4.1. Noções básicas de SQL

O que é a SQL?

SQL é o acrónimo de *Structured Query Language*, ou a tradução literal para português, linguagem de consulta estruturada. A SQL é uma linguagem de programação padrão especificamente criada para armazenar, recuperar, gerir ou manipular dados de um Sistema de Gestão de Bases de Dados Relacionais (SGBDR). Uma Base de Dados Relacional é um conjunto de dados com uma relação pré-definida entre eles. Estes itens estão organizados num conjunto de tabelas com colunas e linhas.

A SQL tornou-se um padrão ISO em 1987.

Esta linguagem de programação é a linguagem de bases de dados mais implementada e é suportada por sistemas relacionais de bases de dados populares, tais como MySQL, SQL Server, and Oracle. A SQL foi desenvolvida pela IBM no início dos anos 1970 e, originalmente, o seu nome era SEQUEL (Structured English Query Language), que mais tarde mudou para SQL.

Usos da SQL

A SQL é uma das linguagens de consulta mais usadas em bases de dados. Alguns dos seus diversos usos são:

- Permitir aos utilizadores aceder aos dados nos sistemas de gestão de bases de dados relacionais;
- Permitir aos utilizadores descrever os dados;
- Permitir aos utilizadores definir e manipular os dados de uma base de dados.

A síntaxe da SQL

As instruções de SQL são diretas, como inglês simples, mas têm uma síntaxe específica.

Uma instrução de SQL é composta por uma sequência de palavras-chave, identificadores, etc., e finalizada com uma semi-vírgula (;).

Exemplo:
SELECT emp_name, hire_date, salary FROM employees WHERE salary > 5000;

Para melhor legibilidade, a mesma instrução pode ser escrita da seguinte forma:

SELECT emp_name, hire_date, salary
FROM employees
WHERE salary > 5000;

No final de cada instrução de SQL deve ser usada uma semi-vírgula (;) — este sinal de pontuação indica o final da instrução ou submete a informação ao servidor da base de dados.

O uso de maiúsculas e minúsculas em SQL

Considere a seguinte instrução de SQL que faculta registos de uma tabela “Employees”:

SELECT emp_name, hire_date, salary FROM employees;

A mesma instrução pode também ser escrita da seguinte forma:

select emp_name, hire_date, salary from employees;

O uso de letras maiúsculas ou minúsculas é indiferente em SQL, o que significa que ‘SELECT’ é o mesmo que ‘select’.

Contudo, as bases de dados e nomes de tabelas podem ser sensíveis ao uso de maiúsculas e minúsculas dependendo do sistema operativo. De maneira geral, as plataformas Unix e Linux são sensíveis a maiúsculas e minúsculas, e as plataformas Windows não.

O comando ‘Select’ em SQL

O comando ‘SELECT’ seleciona ou obtém os dados de uma ou mais tabelas. Este comando pode ser usado para obter a informação de todas as linhas de uma tabela de uma vez só, ou para selecionar apenas as linhas que correspondem a uma condição específica ou a uma combinação de condições.

Suponhamos que temos na nossa base de dados uma tabela com o nome “Employees” (“Funcionários”) e que contém os seguintes registos:

```
| emp_id| emp_name| hire_date| salary| dept_id|

+--------+-------------+-----------+--------+---------+

|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |

|      2 | Tony Montana | 2002-07-15 |   6500 |      1 |

|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |

|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |

|      5 | Martin Blank | 2008-06-24 |   5600 |    NULL |

+--------+-------------+-----------+--------+---------+
```

Selecionar todos os dados da tabela

Para obter os dados de todas as linhas da tabela, deverá ser usada a seguinte instrução:

>> SELECT * FROM employees;

Selecionar colunas específicas da tabela

Se não necessitarmos de todos os dados da tabela, podemos selcionar colunas específicas através da seguinte instrução:

```
SELECT emp_id, emp_name, hire_date, salary
FROM employees;
```

Após executar as instruções supramencionadas, obteremos o seguinte *output*:

| emp_id| emp_name| hire_date| salary|

```
+--------+-------------+-----------+--------+
|      1 | Ethan Hunt   | 1995-10-30 |   5000 |
|      2 | Tony Montana | 1990-07-15 |   6500 |
|      3 | Sarah Connor | 2011-04-13 |   5600 |
|      4 | Rick Deckard | 2005-10-18 |   7200 |
|      5 | Martin Blank | 1996-05-24 |   8000 |
+--------+-------------+-----------+--------+
```

O comando 'SELECT DISTINCT' em SQL

O comando 'SELECT DISTINCT' omite valores duplicados numa consulta. É possível encontrar valores duplicados numa tabela, mas, por vezes, queremos consultar apenas valores únicos.

Síntaxe:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Nos exemplos seguintes, iremos usar a tabela “Customers”, que contém dados sobre os nossos clientes.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

*Tabela 1 – Exemplo de uso do comando ‘SELECT DISTINCT’ com a tabela “Customers” (**Fonte**: https://www.w3schools.com/sql/sql_distinct.asp)*

Para selecionar todos os valores da coluna “Country” da tabela “Customers”, deve ser usada a seguinte instrução:

```
SELECT Country FROM Customers;
```

Contudo, esta instrução incluiría valores duplicados.

De forma a omitir os valores duplicados, deverá ser usado o commando ‘SELECT DISTINCT’:

```
SELECT DISTINCT Country FROM Customers;
```

Suponhamos agora que queremos listar os diferentes países clientes, usaríamos a seguinte instrução:

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Note-se que este exemplo não funciona no Firefox, pois o comando ‘COUNT(DISTINCT column_name)’ não é suportado em MS Access.

Para obter um resultado equivalente em Access, poderá ser usada a seguinte instrução:

```
>> SELECT Count(*) AS DistinctCountries
    FROM (SELECT DISTINCT Country FROM Customers);
```

O comando ‘WHERE’ em SQL

Aprendemos, anteriormente, como obter todos os registos de uma tabela ou colunas de uma tabela.

Contudo, em casos reais, normalmente é necessário selecionar, atualizar ou apagar apenas os registos que correspondem a determinadas condições, como por exemplo utilizadores que pertencem a uma determinada faixa etária, país, etc.

O comando ‘WHERE’ é usado com ‘select, ‘update’ e ‘DELETE’.
Este comando é usado com ‘SELECT’ para extrair apenas os registos que correspondem a determinadas condições.

A síntaxe base é a seguinte:

```
SELECT column_list
FROM table_name
WHERE condition;
```

Analisemos agora alguns exemplos demonstrativos.

Vamos supor que temos na nossa base de dados a tabela “Employees” com os seguintes registos:

```
| emp_id| emp_name| hire_date| salary| dept_id|
+--------+-------------+-----------+--------+---------+
|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |
+--------+-------------+-----------+--------+---------+
```

A seguinte instrução SQL apresentará como resultados todos os funcionários com um salário superior a 7000 que estão presentes na tabela:

```
SELECT * FROM employees WHERE salary > 7000;
```

O comando ‘WHERE’ apenas filtra e exclui os dados indesejados.

Outro exemplo seria selecionar todos os funcionários com o valor ‘department id=1’:

```
SELECT * FROM employees WHERE dept_id=1;
```

A tabela em baixo faculta a lista de operadores que podem ser usados com o comando ‘WHERE’:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Tabela 2 -*Tabela “Operators” com o comando ‘WHERE’ (**Fonte**: https://www.w3schools.com/sql/sql_where.asp)*

Os operadores ‘AND’, ‘OR’ e ‘NOT’ em SQL

O comando ‘WHERE’ pode ser usado com os operadores ‘AND’, ‘OR’, e ‘NOT’.

Os operadores ‘AND’ e ‘OR’ são usados para filtrar registos tendo por base mais do que uma condição.

O operador ‘AND’ exibe um registo caso as condições envolvidas sejam verdadeiras.

A síntaxe 'AND':

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Exemplo: Selecionar todos os campos da tabela "Customers" em que o valores em "Country" e "City" sejam "Germany" e "Berlin", respetivamente.

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

O operador 'OR' exibe um registo caso as condições envolvidas sejam verdadeiras.

A síntaxe 'OR':

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Exemplo 1: Selecionar todos os campos da tabela "Customers" em que o valor em "City" seja "Berlin" ou "München".

```
SELECT * FROM Customers.
WHERE City='Berlin' OR City='München';
```

Exemplo 2: Selecionar todos os campos da tabela "Customers" em que o valor em "Country" seja "Germany" ou "Spain".

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

O operador ‘NOT’ exibe um registo caso as condições envolvidas não sejam verdade.

A síntaxe ‘NOT’:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Exemplo: Selecionar todos os campos da tabela “Customers” em que o valor no campo “Country” não seja “Germany”.

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Conjugar os operadores ‘AND’, ‘OR’ e ‘NOT’

Exemplo 1: Selecionar todas as linhas da tabela “Customers” nas quais o campo “Country” seja “Germany” e o campo ‘City’ seja “Berlin” ou “München”.

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

Exemplo 2: Selecionar todas as linhas da tabela “Customers” nas quais o campo “Country” não seja “Germany” nem “USA”.

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

O comando ‘ORDER BY’ em SQL

O comando ‘ORDER BY’ apresenta os resultados requeridos por ordem crescente ou decrescente. De forma a organizar os resultados por ordem decrescente, use ‘DESC’.

A síntaxe ‘ORDER BY’:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Nos exemplos seguintes será usada a tabela “Customers” apresentada em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 3 – *Tabela “Customers” com um exemplo de uso do comando ‘ORDER BY’* ***(Fonte:*** https://www.w3schools.com/sql/sql_orderby.asp*)*

Exemplo 1: Selecionar todos os clientes da tabela “Customers” e organizá-los por coluna “Country”.

```
SELECT * FROM Customers
ORDER BY Country;
```

Exemplo 2: Selecionar todos os clientes da tabela e organizá-los por coluna “Country” em ordem decrescente.

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Exemplo 3: Selecionar todos os clientes da mesma tabela e organizá-los por “Country” e “Customer Name”.

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

Neste exemplo, os dados são, inicialmente, organizados por “Country”. Contudo, se existirem linhas com o mesmo valor no campo “Country”, estas serão organizadas por “Customer Name”.

Exemplo 4: Selecionar todos os clientes da mesma tabela e organizar os dados por “Country” em ordem crescente, e por “Customer Name” em ordem decrescente.

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

O comando ‘INSERT INTO’ em SQL

O comando ‘INSERT INTO’ insere novos registos numa tabela.

De forma a que o Código seja processado corretamente, os nomes das colunas e os valores a ser inseridos deverão ser especificados.

Síntaxe:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

Tabela 5 – *Exemplo do comando 'INSERT INTO'* **(*Fonte:*** https://www.w3schools.com/sql/sql_insert.asp*)*

Por exemplo, para acrescentar um novo registo à tabela "Customers", use a seguinte instrução:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

Valores 'NULL' em SQL

Um campo com um valor 'NULL' é um campo sem um valor. Se um campo de uma tabela for opcional, é possível inserir um novo registo ou atualizar um registo sem acrescentar um novo valor a este campo.

Nota: Um valor 'NULL' é diferente de um valor zero ou de um campo que contém espaços. Um campo com um valor 'NULL' é um campo que foi **deixado em branco** durante a criação do registo!

Teste de valores 'NULL'

É impossível testar para obter valores 'NULL' com operadores de comparação, como '=', '<' ou '<>'.

Em vez destes operadores, teremos de usar os operadores **'IS NULL'** e **'IS NOT NULL'**.

A Síntaxe 'IS NULL':

```
SELECT column_names
```

```
FROM table_name
WHERE column_name IS NULL;
```

A Síntaxe 'IS NOT NULL':

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Os exemplos seguintes usam a tabela em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 6 – *Tabela "Customers" em uso num exemplo com valores 'NULL'.* ***(Fonte:*** https://www.w3schools.com/sql/sql_null_values.asp*)*

Exemplo de 'IS NULL'

Seleciona todos os clientes com valores 'NULL' (ex., valores vazios) na coluna 'Address':

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Para procurar valores 'NULL', devemos usar sempre 'IS NULL'.

Exemplo de 'IS NULL'

Seleciona todos os clientes com valores 'NOT NULL' (ex., valores que não estejam vazios) na coluna 'Address':

```
SELECT CustomerName, ContactName, Address
```

```
FROM Customers
WHERE Address IS NOT NULL;
```

O comando ‘UPDATE’ em SQL

O comando ‘UPDATE’ altera os registos existentes de uma tabela.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

É necessário algum cuidado ao atualizar os registos de uma tabela. Note-se o uso do comando ‘WHERE’ na instrução ‘UPDATE’. O comando ‘WHERE’ especifíca que registos devem ser atualizados.

Se o comando ‘WHERE’ for **omitido**, todos os registos na tabela serão atualizados!

Exemplo

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujilo Emparedados y helados	Ana Trujilo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 7 – *Tabela “Customers” em uso num exemplo do comando ‘UPDATE’* ***(Fonte:*** https://www.w3schools.com/sql/sql_update.asp)

Para atualizar ‘CustomerID=1’ da tabela “Customers”, utilize a seguinte instrução:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
```

```
WHERE CustomerID = 1;
```

Após a inserção da instrução anterior, a tabela “Customers” terá este aspeto:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 8 – *Tabela “Customers” com um exemplo do uso de ‘UPDATE’* ***(Source:*** https://www.w3schools.com/sql/sql_update.asp*)*

O comando ‘WHERE’ determina quantos registos serão atualizados.

A seguinte instrução SQL atualizará o campo “ContactName” para “Juan” em todos os registos nos quais o campo “Country” tem o valor “Mexico”:

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

O comando ‘DELETE’ em SQL

O comando ‘DELETE’ apaga os registos da tabela existentes.

```
DELETE FROM table_name WHERE condition;
```

Note-se que é necessária alguma caução aquando apagar registos da tabela! Repare no uso do comando 'WHERE' na instrução 'DELETE'. O comando 'WHERE' determina que registo(s) deve(m) ser eliminado(s).

Se o comando 'WHERE' for **omitido**, todos os registos da tabela serão eliminados!

Exemplo:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Apagar todos os registos

É possível eliminar todas as linhas de uma tabela sem eliminar a tabela. Quer isto dizer que a estrutura, os atributos e índices da tabela ficarão intactos:

```
DELETE FROM table_name;
```

O comando 'SELECT TOP' em SQL

O comando 'SELECT TOP' é usado para determinar o número de registos que serão acedidos.

Este comando é útil para tabelas grandes com milhares de registos. No entanto, exibir um grande número de registos pode afetar o desempenho.

Note-se que nem todos os sistemas de bases de dados suportam o comando 'SELECT TOP'. O MYSQL suporta o comando 'LIMIT' para selecionar um número limitado de registos, ao passo que o Oracle usa 'FETCH FIRST n ROWS ONLY' e 'ROWNUM'.

SQL Server /Síntaxe MS Access:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

A Síntaxe MySQL:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

A Síntaxe Oracle 12:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

A Síntaxe Older Oracle:

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

A Síntaxe Older Oracle Syntax com ORDER BY:

```
SELECT *
FROM (SELECT column_name(s) FROM table_name ORDER BY column_name(s))
WHERE ROWNUM <= number;
```

Para os exemplos seguintes será usada a tabela “Customers” em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

*Tabela 9 – Tabela "Customers" com exemplos de 'SELECT TOP' (**Fonte:** https://www.w3schools.com/sql/sql_top.asp)*

Exemplos de 'TOP', 'LIMIT' e 'FETCH FIRST'

Use a seguinte instrução para selecionar os primeiros três registos da tabela 'Customers' em SQLServer/MS Access:

```
SELECT TOP 3 * FROM Customers;
```

Para executar uma consulta com o mesmo resultado que em cima mas em MySQL, use a seguinte instrução:

```
SELECT * FROM Customers
LIMIT 3;
```

Para executar uma consulta com o mesmo resultado em Oracle:

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

Exemplos de 'TOP PERCENT'

Para selecionar 50% dos registos da tabela "Customers", execute a seguinte instrução em SQL Server/MS Access:

```
SELECT TOP 50 PERCENT * FROM Customers;
```

A seguinte instrução equivale à de cima em Oracle:

```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

Exemplos do comando 'ADD a WHERE'

No exemplo seguinte, iremos selecionar os primeiros três registos da tabela “Customers”, em que o campo “Country” seja o valor “Germany” para SQL Server/MS Access:

```
>> SELECT TOP 3 * FROM Customers
    WHERE Country='Germany';
```

A instrução equivalente em MySQL:

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

A instrução equivalente em Oracle:

```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

A função ‘MIN’ e ‘MAX’ em SQL

A função ‘MIN()’ apresenta os valores mais pequenos das colunas selecionadas.

A Síntaxe ‘MIN()’

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

A função ‘MAX()’ apreesenta os valores mais altos das colunas selecionadas.

A Síntaxe ‘MAX()’

```
SELECT MAX(column_name)
```

```
FROM table_name
WHERE condition;
```

Nos exemplos seguintes, será usada a tabela “Products” que segue em baixo:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Tabela 10 - *Tabela “Products” com exemplos de ‘MIN’ e ‘MAX’* ***(Fonte:***
https://www.w3schools.com/sql/sql_min_max.asp*)*

Exemplo 1: Para encontrar o preço do produto mais barato:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Exemplo 2: Para encontrar o preço do produto mais caro:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

As funções ‘COUNT’, ‘AVG’ E ‘SUM’ EM SQL

A função ‘COUNT’ apresenta o número de linhas que correspondem a critérios específicos.

A Síntaxe ‘COUNT()’

```
SELECT COUNT(column_name)
FROM table_name
```

```
WHERE condition;
```

A função ‘AVG()’ apresenta o valor médio de uma coluna numérica.

A Síntaxe ‘AVG()’

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

A função ‘SUM()’ apresenta a soma total de uma coluna numérica.

A Síntaxe ‘SUM()’

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Para os exemplos seguintes será usada a tabela “Products” em baixo:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Tabela 11 – *Tabela “Products” com exemplos de ‘Count’, ‘Avg’, ‘Sum’* ***(Fonte:*** *https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

Exemplo de ‘COUNT()’

Para executar uma consulta para enocntrar o número de produtos:

```
SELECT COUNT(ProductID)
FROM Products;
```

Exemplo de ‘AVG()’

Para executar uma pesquisa para encontrar o preço médio de todos os produtos:

```
SELECT AVG(Price)
FROM Products;
```

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

***Tabela 12** – Tabela "OrdersDetails" com exemplos 'Count', 'Avg', 'Sum' **(Fonte:** https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

No exemplo seguinte, será usada a tabela "OrdersDetails", apresentada em cima, para procurer o total de 'Quantity':

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

O operador 'LIKE' em SQL

O operador 'LIKE' é usado numa instrução 'WHERE' para pesquisar um padrão específico numa coluna.

A Síntaxe 'LIKE'

```
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

Em baixo estão alguns exemplos de diferentes operadores 'LIKE' com carateres '%' e '_':

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Tabela 13 *– Operadores 'LIKE' (**Fonte**: https://www.w3schools.com/sql/sql_like.asp)*

Veremos agor alguns exemplos com a tabela "Customers" que segue em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Tabela 14 *– Tabela "Customers" com exemplos 'LIKE' (**Fonte:** https://www.w3schools.com/sql/sql_like.asp)*

Exemplo 1: Para selecionar todos os clientes com nome a começar por 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Exemplo 2: Para selecionar todos os clientes cujo nome termina em 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Exemplo 3: Para selecionar todos os clientes cujo nome contém 'or' em qualquer posição:

```
SELECT * FROM Customers
```

```
WHERE CustomerName LIKE '%or%';
```

Exemplo 4: Para selecionar todos os nomes de clientes que têm ‘r’ como segunda letra:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Exemplo 5: Para selecionar todos os nomes de clientes que comecem com ‘a’ e tenham pelo menos três carateres:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Exemplo 6: Para selecionar os nomes de clientes que comecem com ‘a’ e terminem em ‘o’:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%o';
```

Exemplo 7: Para selecionar os nomes de clientes que não comecem por ‘a’:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Wildcards em SQL

Um *wildcard* substitui um ou mais carateres numa cadeia. É usado com o operador ‘LIKE’.

O operador ‘LIKE’ também é usado com o comando ‘WHERE’ para pesquisar um padrão específico numa coluna, tal como vimos na subsecção anterior.

Wildcards em MS Access

Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

*Tablela 15 – 'Wildcards' em Access (**Fonte:** https://www.w3schools.com/sql/sql_wildcards.asp)*

Wildcards no SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

*Tabela 16 – 'Wildcards' em SQL Server (**Fonte**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Os *wildcards* podem ser combinados, considere o exemplo em baixo:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

*Tabela 17 - Exemplos de 'Wildcards' com '%' and '_' (**Fonte**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Consideremos agora alguns exemplos com a tabela "Customers":

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujilo Emparedados y helados	Ana Trujilo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK

Tabela 18 – *Tabela "Customers" com exemplos de 'WILDCARDS' (**Fonte**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Exemplos com o *wildcard* '%'

Neste exemplo, selecionamos todos os clientes com uma "City" a começar por "ber":

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

No exemplo seguinte, selecionamos todos os clientes com uma "City" que contenha "es":

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

Exemplos com o *wildcard* '_'

Neste exemplo, são selecionados todos os clientes com uma "City" que se inicie por qualquer letra seguida de "ondon":

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

Para selecionar todos os clientes com uma "City" a iniciar-se com a letra 'L', seguida por qualquer carater seguido por "on":

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Exemplos com o *wildcard* '[charlist]'

Para selecionar todos os clientes com uma "City" a iniciar com 'b', 's' ou 'p':

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

No exemplo seguinte, selecionaremos todos os clientes com uma "City" a começar com 'a', 'b', ou 'c':

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Exemplos com o *wildcard* '[!charlist]'

O ponto de exclamação mostra carateres que não contêm uma cadeia específica. Por exemplo, queremos selecionar todos os clientes com uma "City" que não começa por 'b', 's' ou 'p':

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

Em alternativa, podemos usar a seguinte instrução:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

O operador 'IN' em SQL

O operador 'IN' é usado para especificar valores múltiplos numa instrução 'WHERE'. Pode considerar-se que satisfaz várias condições.

Síntaxe 1:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Síntaxe 2:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Há duas formas de usar o operador 'IN', como vimos anteriormente.

Suponhamos que temos uma tabela "Customers" que contém as seguintes colunas: "CustomerID", "CustomerName", "ContactName", "Address", "City", "PostalCode" and "Country".

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain

***Tabela 19** – Tabela "Customers" com exemplos de operador 'IN' (**Fonte:** https://www.w3schools.com/sql/sql_in.asp)*

Como exemplo, queremos selecionar todos os clientes localizados na Alemanha ('Germany'), França ('France') ou Reino Unido ('UK'):

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK')
```

Outro exemplo, é selecionar todos os clientes que **não** estão localizados na Alemanha (‘Germany’), França (‘France’) ou Reino Unido (‘UK’):

```
SELECT * FROM Customers
WHERE Country NOT IN ( ‘Germany’,  ‘France’, ‘UK’)
```

Consideremos ainda um terceiro exemplo no qual queremos selecionar os clientes que são do mesmo país que os fornecedores (*suppliers*):

```
SELECT * FROM Customers
WHERE Country  IN (SELECT Country FROM Suppliers)
```

O operador ‘BETWEEN’ em SQL

O operador ‘BETWEEN’ faculta um conjunto de valores dos quais escolher. Os valores podem ser texto, números ou datas. O operador ‘BETWEEN’ inclui os valores iniciais e finais.

Síntaxe:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Consideremos a seguinte tabela, que contém informação sobre diferentes produtos:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

Tabela 20 *– Exemplo de operador ‘BETWEEN’ (**Fonte**: https://www.w3schools.com/sql/sql_between.asp)*

Seguem vários exemplos com os seguintes operadores: 'BETWEEN', 'NOT BETWEEN', 'BETWEEN' com 'IN', 'BETWEEN' e 'NOT BETWEEN' com valores em texto e datas 'BETWEEN'.

Exemplo de 'BETWEEN': Para selecionar todos os produtos com um intervalo de preço entre 10 e 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Exemplo de 'NOT BETWEEN': Exibe todos os produtos fora do intervalo definido no exemplo anterior:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

Exemplo de 'BETWEEN' com 'IN': Seleciona todos os produtos com um intervalo de preços entre 10 e 20 e não mostra os produtos com a "CategoryID" 1, 2 ou 3:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

Exemplos de 'BETWEEN' com valores em texto: Seleciona todos os produtos com um "ProductName" entre "Carnarvorn Tigers" e "Mozzarella di Giovanni":

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Exemplos de 'NOT BETWEEN' com valores em texto: Seleciona todos os produtos com um "ProductName" que não esteja entre "Carnarvorn Tigers" e "Mozzarella di Giovanni":

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Consideremos agora a seguinte tabela com informação relativa a diferentes “Orders” (encomendas):

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/9/1996	1
10252	76	4	7/10/1996	2

***Tabela 21** – Exemplo de operador ‘BETWEEN’ (**Fonte**: https://www.w3schools.com/sql/sql_between.asp)*

Exemplo de datas ‘BETWEEN’: Selecionar todas as encomendas com data entre ’01-July-1996’ e ’31-July-1996’.
Isto pode ser feito de duas formas, usando um cardinal (#) ou aspas (“”):

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

OU

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Aliases em SQL

Os aliases atribuem um nome temporário a uma tabela ou coluna de uma tabela. Um alias existe apenas durante uma pesquisa e é, normalmente, usado para tornar os nomes das colunas mais legíveis. Um aliasé criado através do uso da palavra-chave **AS.**

Síntaxa para colunas alias:

```
SELECT column_name AS alias_name
FROM table_name;
```

Síntaxe para tabelas alias:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Colunas aliases

Analisemos um exemplo que cria dois aliases, um para cada coluna:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Outro exemplo para criar dois aliases, novamente:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

Note-se que este é colocado entre parêntesis retangulares ([ ]) porque o alias contém espaços. As aspas podem ser usadas em alternativa aos parêntesis.

Podemos também criar um alias com uma ou mais colunas, consideremos o exemplo em baixo:

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address
FROM Customers;
```

A instrução em cima citada difere em MySQL:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address
FROM Customers;
```

Tabelas Aliases

O exemplo seguinte seleciona todas as encomendas da tabela de clientes com "CustomerID=4" ("Around the Horn").

Neste exemplo, são usados aliases para encurtar a pesquisa:

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

Uma pesquisa em aliases assemelhar-se-á a isto:

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;
```

A operação 'JOIN' em SQL

A operação 'JOIN' combina linhas de duas ou mais tabelas tendo por base uma coluna comum em ambas as tabelas.

Consideremos as tabelas "Orders" e "Customers":

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

***Tabelas 22 & 23** – Tabelas "Orders" e "Customers" num exemplo 'JOIN' (**Fonte:** https://www.w3schools.com/sql/sql_join.asp)*

Ao analisar as duas tabelas é possóvel verificar que estas têm uma coluna em comum, "CustomerID". Com base nesta coluna comum, poderemos criar uma instrução SQL com 'INNER JOIN', que seleciona registos com valores correspondentes em ambas as tabelas.

Exemplo:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Este comando criará algo semelhante ao que consta na tabela em baixo:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

***Tabela 24** – Exemplo 'JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join.asp)*

Existem quatro junções ('joins') diferentes em SQL:

1. **(INNER) JOIN:** exibe registos com valores equivalentes e mambas as tabelas;
2. **LEFT (OUTER) JOIN:** apresenta todos os registos da tabela da esquerda e os registos correspondentes da tabela da direita;
3. **RIGHT (OUTER) JOIN:** apresenta todos os registos da tabela da direita e os valores correspondentes da tabela da esquerda;

4. **FULL (OUTER) JOIN:** exibe todos os registos quando há uma correspondência na tabela da esquerda ou da direira.

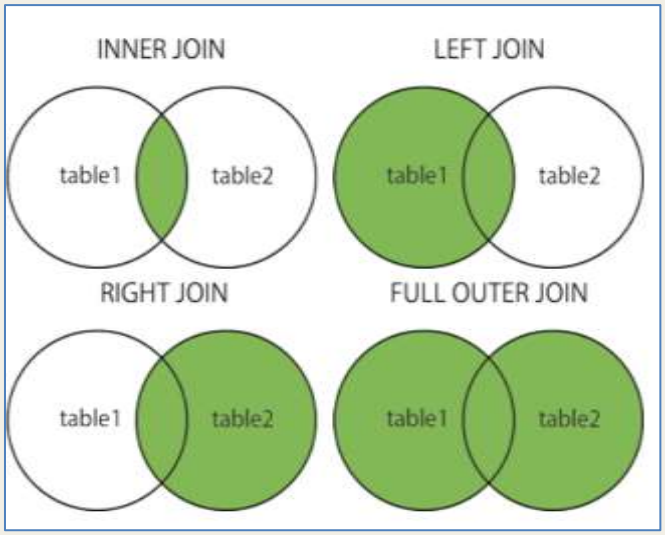

***Figura 1** – Diferentes tipos de 'JOINS' (**Fonte**: https://www.w3schools.com/sql/sql_join.asp)*

'INNER JOIN' em SQL

A operação 'INNER JOIN' seleciona registos que têm valores equivalentes em ambas as tabelas.

Síntaxe:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Para este exemplo foram usadas as mesmas tabelas que na subsecção anterior.

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

***Tabelas 25 & 26** – Tabelas "Orders" e "Customers" em exemplo 'JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join_inner.asp)*

Neste exemplo, queremos obter os nomes dos clientes e as suas "Order IDs" respetivas:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note-se que a instrução 'INNER JOIN' selecionará todas as linhas correspondentes de ambas as tabelas. Se os registos da tabela "Orders" não tiverem registos equivalentes na tabela "Customers", não serão selecionados.

No exemplo seguinte, veremos como juntar três tabelas com dados referentes a clientes e informação de envio:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

'LEFT JOIN' em SQL

A operação 'LEFT JOIN' exibe todos os registos da tabela da esquerda e os registso equivalentes da tabela da direita. Se não forem encontrados dados correspondentes, não serão apresentados quaisquer registos da tabela da direita como resultado.

Síntaxe:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
```

```
ON table1.column_name = table2.column_name;
```

Atenção, a operação ‘LEFT JOIN’ denomina-se por ‘LEFT OUTER JOIN’ em algumas bases de dados.

Como exemplo, vamos selecionar todos os clientes e quaisquer encomendas que estes possam ter feito:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note-se que todos os registos da tabela da esquerda,”Customers”, serão exibidos, mesmo que não existam registos correspondentes na tabela da direita, “Orders”.

‘RIGHT JOIN’ em SQL

A operação ‘RIGHT JOIN’ segue a mesma lógica descrita anteriormente, mas do lado direito em vez do lado esquerdo.

Esta operação exibe todos os registos da tabela da direita e os registos correspondentes da tabela da esquerda, se estes existirem.

Síntaxe:

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Consideremos as seguintes tabelas, 'Orders' e 'Employees':

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

***Tabela 28** – Tabela "Orders" com um exemplo 'RIGHT JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join_right.asp)*

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

***Tabela 29** – Tabela "Employees" com um exemplo 'RIGHT JOIN' (**Fonte:** https://www.w3schools.com/sql/sql_join_right.asp)*

O exemplo seguinte exibirá todos os funcionários e quaisquer encomendas que possam ter feito:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note-se que a operação 'RIGHT JOIN' exibirá todos os registos da tabela da direita, "Employees", mesmo que não exista correspondência com a tabela da esquerda, "Orders". Aplica-se a mesma lógica que vimos anteriormente na operação 'LEFT JOIN'.

'FULL JOIN' em SQL

A operação 'FULL JOIN' recupera todos os registos quando são encontrados registos correspondentes quern a tabela da direita, quern a tabela da esquerda.

Note-se que 'FULL OUTER JOIN' e 'FULL JOIN' são a mesma coisa e 'FULL JOIN' pode exibir um número maior de resultados.

Síntaxe:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Consideremos as seguintes tabelas, a tabela "Orders" e a tabela "Customers":

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Tabela 30 *– Tabela "Orders" com um exemplo 'FULL JOIN' (**Fonte:** https://www.w3schools.com/sql/sql_join_full.asp)*

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Tabela 31 *– Tabela "Customers" com um exemplo 'FULL JOIN' (**Fonte**: https://www.w3schools.com/sql/sql_join_full.asp)*

Um exemplo que seleciona todos os clientes (*customer*s) e encomendas (*orders*):

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

O resultado deste 'FULL JOIN' assemelhar-se-á a algo como isto:

CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

***Tabela 32** – Exemplo de 'FULL JOIN' com tabelas "Customers" e "Orders" (**Fonte**: https://www.w3schools.com/sql/sql_join_full.asp)*

Aqui podemos verificar que esta operação exibe todos os resultados correspondentes de ambas as tabelas, mesmo que estes não sejam encontrados. Neste caso, é atribuído o valor *nul*l.

'SELFJOIN' em SQL

Um 'SELF JOIN' é considerado uma operação 'JOIN' normal, mas a tabela é incluída também.

Síntaxe:

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

'T1' e 'T2' são aliases usados para a mesma tabela.

Consideremos a tabela "Customers" como exemplo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

*Tabela 33 – Exemplo de 'SELF JOIN' em tabela 'Customers' (**Fonte**: https://www.w3schools.com/sql/sql_join_self.asp)*

Neste exemplo, queremos selecionar clientes da mesma cidade:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

'UNION' em SQL

O operador 'UNION' é usado para combiner o conjunto de resultados de duas ou mais instruções 'SELECT'.

Há alguns requisitos para viabilizar um 'UNION':

1. Ao utilizar o operador 'UNION', todas as instruções 'SELECT' devem ter o mesmo número de colunas;
2. As colunas devem ter tipos de dados semelhantes;
3. As colunas devem estar na mesma ordem em todas as instruções 'SELECT'.

Síntaxe:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

* Note-se que o operador 'UNION' seleciona apenas valores diferentes por defeito.

Para permitir valores duplicados, use 'UNION ALL':

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

* Note-se que os nomes das colunas são normalmente iguais nas duas instruções 'SELECT'.

Vejamos agor alguns exemplos de 'UNION', 'UNION ALL' e 'UNION' com instruções 'WHERE' para podermos compreender melhor como usá-los.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Tabela 34 *– Exemplo 'UNION' com tabela "Customers" (**Fonte:** https://www.w3schools.com/sql/sql_union.asp)*

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

Tabela 35 *– Sexemplo 'UNION' com tabela "Suppliers" (**Fonte:** https://www.w3schools.com/sql/sql_union.asp)*

O primeiro exemplo é usado para obter cidades diferentes de ambas as tabelas exibidas em cima:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Por estarmos a usar 'UNION', os fornecedores (*suppliers*) da mesma cidade serão listados apenas uma vez. Se quisermos ver os valores duplicados, devemos usar 'UNION ALL'.

O exemplo seguinte apresentará como resultado os valores duplicados de ambas as tabelas:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

Neste exemplo, através do comando 'WHERE' serão exibidas as diferentes cidades alemãs presents nas tabelas "Customers" e "Suppliers":

```
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
UNION
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
ORDER BY City;
```

Este exemplo é semelhante ao anterior, mas, possivelmente, serão exibidos valores duplicados:

```
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
UNION ALL
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
ORDER BY City;
```

O próximo exemplo lista todos os clientes e fornecedores:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
```

```
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

Repare que neste caso usamos ‘AS’ para criar um alias para a pesquisa que irá desaparecer assim que esta fique concluída.

‘GROUP BY’ em SQL

A instrução ‘GROUP BY’ agrupa linhas com os mesmos valores em linhas resumidas. Suponhamos, por exemplo, que queremos consultar o número de clientes de cada país. Esta instrução é também frequentemente usada com funções combinadas, tais como ‘COUNT()’, ‘MAX()’, ‘MIN()’, ‘SUM()’, ‘AVG()’, para agrupar os resultados em uma ou mais colunas.

Síntaxe:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Para os exemplos seguintes, iremos usar a tabela “Customers” em baixo:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Tabela 36 *– Exemplo de ‘GROUP BY’ com tabela “Customers” (**Fonte:** https://www.w3schools.com/sql/sql_groupby.asp)*

Neste primeiro exemplo, iremos listar o número de clientes de cada país:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

No segundo exemplo, iremos listar novamente o número de clientes em cada país, mas por ordem decrescente:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
ORDER BY COUNT(CustomerID) DESC;
```

No exemplo seguinte, iremos usar 'GROUP BY' com 'JOIN' com as tabelas "Orders" e "Shippers":

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

Tabela 37 *– Exemplo 'GROUP BY' com tabela "Orders" (**Fonte**: https://www.w3schools.com/sql/sql_groupby.asp)*

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

Tabela 38 *– Exemplo de 'GROUP BY' com tabela "Shippers" (**Fonte:** https://www.w3schools.com/sql/sql_groupby.asp)*

Neste exemplo, iremos listar o número de encomendas (*orders*) enviados por cada exportador (*shipper*):

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
```

```
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Note-se que começamos por selecionar os nomes dos exportadores (*shippers*) na tabela “Shippers” e contar as encomendas tendo por base os “OrderID” guardados como um alias.

Em seguida, executamos um ‘LEFT JOIN’ para juntar a tabela “Shippers” (table 2) com a tabela “Orders” (tabela 2) e agrupamo-las por nome de exportador.

‘HAVING’ em SQL

A instrução ‘HAVING’ foi adicionada à SQL porque o comando ‘WHERE’ não pode ser usado com funções combinadas.

Síntaxe:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Tabela 39** – Exemplo de ‘HAVING’ com tabela “Customers” (**Fonte:** https://www.w3schools.com/sql/sql_having.asp)*

Para este exemplo, usaremos novamente a tabela “Customers”. Neste caso, queremos listar o número de clientes de cada país, mas também queremos incluir países que têm mais do que cinco clientes.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Neste exemplo, queremos listar o número de clientes por país, novamente, e incluir os países com mais de cinco clientes por ordem decrescente.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

Tentemos mais alguns exemplos para combiner o que aprendemos até agora.

Nos dois exemplos seguintes iremos usar as tabelas “Orders” e “Employees”:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

Tabela 40 *– Exemplo de ‘HVING’ com tabela “Orders” (**Fonte:** https://www.w3schools.com/sql/sql_having.asp)*

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

Tabela 41 *– Exemplo de ‘HAVING’ com tabela “Orders” (**Fonte:** https://www.w3schools.com/sql/sql_having.asp)*

O exemplo seguinte lista os funcionários que registaram mais de dez encomendas:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

Neste exemplo, iremos listar os funcionários de “Davolio” ou “Fuller” que tenham registado encomendas mais do que 25 vezes:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
WHERE LastName = ‘Davolio’ OR LastName = ‘Fuller’
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

‘SELECT INTO’ em SQL

A instrução ‘SELECT INTO’ copia dados de uma tabela para uma nova tabela.

Síntaxe para copiar todas as colunas para uma nova tabela:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Síntaxe para copiar apenas algumas colunas para uma nova tabela:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
```

```
WHERE condition;
```

A tabela nova irá manter os nomes e tipos das colunas. É possível criar novas colunas com o comando ‘AS’.

Exemplo de como criar uma cópia de segurança da tabela “Customers”:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

Exemplo do uso do comando ‘IN’ para copiar a tabela para uma tabela nova em outra base de dados:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

Exemplo de como copiar apenas algumas colunas para uma tabela nova:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

Exe4mplo de como copiar apenas os clientes alemães para uma tabela nova:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

Exemplo de como copiar dados de várias tabelas para uma tabela nova:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

‘SELECT INTO’ também pode ser usado para criar uma tabela nova e vazia usando o mesmo esquema de outra tabela.

Para tal, temos de acrescentar um comando 'WHERE' que não obtém quaisquer dados:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

'INSERT INTO SELCT' em SQL

A instrição 'INSERT INTO SELECT' copia dados de uma tabela e insere-os em outra tabela. Isto requere que o tipo de dados da fonte correspondam ao tipo de dados da tabela para onde estes serão copiados.

Síntaxe para copiar todas as colunas de uma tabela para outra:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Síntaxe para copiar apenas algumas colunas de uma tabela para outra:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujilo Emparedados y helados	Ana Trujilo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

*Tabela 42 – Exemplo 'INSERT INTO SELECT' com tabela "Customers" (**Fonte:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*

SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

*Tabela 43 – Exemplo de 'INSERT INTO SELECT' com tabela "Suppliers" (**Fonte:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*

Para os exemplos seguintes, usaremos as tabelas "Customers" e "Suppliers".

O primeiro exemplo copia a tabela "Suppliers" para a tabela "Customers" (note-se que as colunas sem dados serão registadas com valores 'null'):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

Este exemplo copia a tabela "Suppliers" para a tabela "Customers" para preencher todas as colunas:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
```

O terceiro exemplo copia apenas os fornecedores alemães para a tabela "Customers":

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

'CASE' em SQL

A instrução 'CASE' repassa uma série de condições e apresenta um valor quando a primeira condição é atendida. Pense nesta instrução como uma instrução 'IF' e depois 'ELSE. Quando uma condição é verificada como verdadeira, esta isntrução pára de

procurar. Se nenhuma condição for dada como verdadeira, esta instrução apresentará como resultado o valor na instrução 'ELSE'.

Note-se que se não existir uma instrução 'ELSE' e nenhuma condição for dada como verdadeira, o resultado será o valor 'NULL'.

Syntax:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

__Tabela 44__ – Exemplo 'CASE' com tabela "Orders" (__Fonte:__ https://www.w3schools.com/sql/sql_case.asp)

Nos exemplos seguintes, iremos usar a tabela "Orders".

O primeiro exemplo analisa uma série de condições e apresenta um valor quando a primeira condição for atendida:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
```

```
END AS QuantityText
FROM OrderDetails;
```

No segundo exemplo, iremos ordenar os clientes por cidade (*city*). Note-se que se o campo “City” for ‘NULL’ será ordenado por país (*country*).

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
    (CASE
        WHEN City IS NULL THEN Country
        ELSE City
END);
```

Funções ‘NULL’ em SQL

As funções ‘NULL’ incluem: ‘IFNULL()’, ‘ISNULL()’, ‘COALESCE()’ e ‘NVL()’.

Para os exemplos seguintes, usaremos a tabela “Products” em baixo:

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Tabela 45 – *Exemplo de funções ‘NULL’ com tabela “Products” (**Fonte:** https://www.w3schools.com/sql/sql_isnull.asp)*

Digamos que a coluna “UnitsOnOrder” é opcional e pode conter valores ‘NULL’.

Exemplo:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)
FROM Products;
```

Aqui podemos ver que se algum dos valores de “UnitsOnOrder” for ‘null’, o resultado também será ‘null’.

Vejamos agora como Podemos ultrapassar este problema.

Em MySQL, Podemos usar a função ‘ISNULL()’ que permite obter um valor alternative se uma expressão for ‘null’:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Ou podemos usar a função ‘COALESCE()’:

```
SELECT ProductName, UnitPrice * (UnitsInStock + COALESCE(UnitsOnOrder, 0))
FROM Products;
```

Em SQL Server, a função ‘ISNULL()’ faz o mesmo que em MySQL:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Em MS Access, a função ‘IsNull()’ apresenta ‘TRUE(-1)’ se a expressão for um valor ‘null’, em caso contrário, apresenta ‘FALSE (0)’:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0, UnitsOnOrder))
FROM Products;
```

Em Oracle, a função ‘NVL()’ faz a mesma coisa:

```
SELECT ProductName, UnitPrice * (UnitsInStock + NVL(UnitsOnOrder, 0))
FROM Products;
```

Comentários em SQL

Os comentários em SQL explicam as seções das isntruções SQL ou previnem a sua execução.

Note-se que os exemplos nesta secção não são suportados por Firefox ou Microsoft Edge, que são bases de dados Microsoft Access. Normalmente, os comentários não são suportados por bases de dados Microsoft Access.

Os comentários de uma linha começam com - - (dois travessões):

```
--Select all:
SELECT * FROM Customers;
```

Ou podem ser usados da seguinte forma para ignorar o fim da linha:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Ou para ignorar uma instrução:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Comentários com linhas múltiplas começam por /* e terminam em */. Qualquer texto escrito entre estes dois símbolos será ignorado.

Exemplo:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

Para ignorar parte de uma instrução também podemos usar /**/.

Exemplo 1:

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Exemplo 2:

```
SELECT * FROM Customers WHERE (CustomerName LIKE 'L%'
OR CustomerName LIKE 'R%' /*OR CustomerName LIKE 'S%'
OR CustomerName LIKE 'T%'*/ OR CustomerName LIKE 'W%')
AND Country='USA'
ORDER BY CustomerName;
```

Operadores SQL

Operadores Aritméticos usados em SQL:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

__Tabela 46__ – Operadores Aritméticos (__Fonte:__ https://www.w3schools.com/sql/sql_operators.asp)

Operadores *Bitwise* usados em SQL:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

__Tabela 47__ – Operadores Bitwise (__Fonte__: https://www.w3schools.com/sql/sql_operators.asp)

Operadores Comparativos usados em SQL:

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

***Tabela 48** – Operadores Comparativos (**Fonte**: https://www.w3schools.com/sql/sql_operators.asp)*

Operadores Compostos usados em SQL:

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

***Tabela 49** - Operadores Compostos (**Fonte:** https://www.w3schools.com/sql/sql_operators.asp)*

Operadores Lógicos usados em SQL:

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Tabela 50 *- Operadores Lógicos (**Fonte**: https://www.w3schools.com/sql/sql_operators.asp)*

4.2. Bases de Dados SQL

Tal como mencionado na secção anterior dedicada às instruções básicas usadas em SQL, esta linguagem de programação é maioritariamente usada em bases de dados relacionais. Nesta secção, iremos aprender como criar, alterar e manipular uma base de dados com SQL.

Iremos começar com instruções simples e avançar progressivamnete para instruções mais complicadas.

'CREATE DATABASE' em SQL

A instrução 'CREATE DATABASE' cria uma nova base de dados SQL.

Síntaxe:
CREATE DATABASE DatabaseName;

Nota: Lembre-se, o nome da base de dados deve ser único no Sistema de Gestão de Base de dados Relacional que está a usar, e garanta que é o administrador do mesmo antes de criar qualquer base de dados.

Imaginemos que queremos criar a base de dados 'testDB', iremos usar a seguinte instrução:

```
CREATE DATABASE testDB;
```

'DROP DB' em SQL

A instrução 'DROP DATABASE' elimina as bases de dados SQL existentes.

```
DROP DATABASE DatabaseName;
```

Antes de apagar a base de dados, garante que não precisa da informação que a mesma contém, porque esta será eliminada na sua totalidade.

Recorda-se da base de dados que criamos, ‘testDB’? Agora vamos apagá-la.

Exemplo:

```
DROP DATABASE testDB;
```

‘BACKUP DB’ em SQL

A instrução ‘BACKUP DATABASE’ faxz uma cópia de segurança complete de uma base de dados SQL existente.

Para usar esta instrução, é necessário facultar duas coisas: o nome da base de dados e a localizaçã do ficheiro.

```
BACKUP DATABASE DatabaseName
TO DISK = ‘filepath’
```

Exemplo:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```

Nota: Para evitar problemas técnicos, é melhor guardar a cópia de segurança numa unidade de disco diferente daquel onde se encontra a base de dados.

Existe outra opção na qual é feita uma cópia de segurança diferenciada com base nas mudanças que foram feitas desde a última cópida de segurança feita. Este tipo de cópia de segurança reduaz o tempo desta operação.

Para tal, use a seguinte síntaxe:

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

Exemplo:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
```

```
WITH DIFFERENTIAL;
```

'CREATE TABLE' em SQL

A instrução 'CREATE TABLE' cria uma nova tabela numa base de dados.

Síntaxe:

```
CREATE TABLE table_name (
        column1 datatype,
        column2 datatype,
        column3 datatype,
        .....
);
```

Nesta instrução, é necessário especificar os **nomes das colunas** e os **tipos de dados** que a coluna irá conter.

Há muitos tipos de dados, como *integer*, *date* ou *varchar*. Dependendo do tipo de dados que quer armazenar, deve escolher a opção mais adequada. Por exemplo, se tem uma coluna com o nome "Date of Birth", então escolheria *date* como tipo de dados.

Exemplo:

```
CREATE TABLE Persons (
        PersonID int,
        LastName varchar(255),
        FirstName varchar(255),
        Address varchar(255),
        City varchar(255)
);
```

Este exemplo criará uma tabela com o nome 'Persons' e com 5 colunas.

A coluna “PersonID” irá conter um *integer* (int); as colunas “LastName”, “FirstName”, “Address” e “City” irão conter carateres com extenão máxima de 255.

A tabela deverá assemelhar-se a este exemplo:

PersonID	LastName	FirstName	Address	City

Tabela 51 *– Exemplo de ‘CREATE TABLE’ com tabela vazia (**Fonte:** https://www.w3schools.com/sql/sql_create_table.asp)*

Também pode criar uma tabela ao usar outra tabela e escolher que colunas quern a tabela nova. Tenha em consideração que os dados da tabela existente serão usados para preencher as entradas da tabela nova.

A síntaxe é a seguinte:

```
CREATE TABLE new_table_name AS
SELECT column1, column2,...
FROM existing_table_name
WHERE ....;
```

Tal como aprendemos na sectção anterior:

- ‘SELECT’ especifica as colunas da tabela existente;
- ‘FROM’ especifica o nome da tabela existente;
- ‘WHERE’ pode ser usado se quisermos um conjunto de registos que preencham uma condição específica.

Exemplo:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

‘DROP TABEL’ em SQL

Similarmente a ‘DROP DATABASE’, esta instrução apaga uma tabela existente numa base de dados.

Lembre-se que deve ter a certeza de que não precisa da informação que consta nessa tabela antes de a apagar.

Síntaxe:

```
DROP TABLE TableName;
```

Exemplo:

```
DROP TABLE Persons;
```

Também pode escolher apagar apenas os dados da atebla e não a tabela em si.

Imagine que cria uma tabela nova a partir de uma tabela existente, a tabela tem a estrutura que pretendia mas quer acrescentar novas entradas. A instrução ‘TRUNCATE TABLE’ é útil num caso destes.

Síntaxe:

```
TRUNCATE TABLE TableName;
```

Exemplo:

```
TRUNCATE TABLE Persons;
```

‘ALTER TABLE’ em SQL

A instrução ‘ALTER TABLE’ pode adicionar, apagar ou alterar colunas de uma tabela existente. Pode também ser usada para acrescentar ou eliminar limitações de uma tabela

Vejamos a síntaxe usada para acrescentar uma coluna:

```
ALTER TABLE TableName
ADD column_name datatype;
```

Esta instrução assemelha-se à forma como criamos uma tabela ao determinar o nome da coluna e o tipo de dados a incluir nessa coluna.

Exemplo:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

Para apagar uma coluna de uma tabela, usamos a instrução 'DROP'.

Lembre-se que alguns sistemas de bases de dados não permitem que os utilizadores apaguem colunas.

Síntaxe:

```
ALTER TABLE TableName
DROP COLUMN ColumnName;
```

Por exemplo, vamos apagar a coluna que criámos:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

Para alterar o tipo de dados de uma coluna, pode usar a seguinte instrução dependendo do SGBD que está a usar:

- 'ALTER COLUMN' (para SQL Server/MS Access);
- 'MODIFY COLUMN' (para My SQL/ Oracle anterior à versão 10G);
- 'MODIFY' (para Oracle version 10G e mais recentes).

Síntaxe:

```
ALTER TABLE TableName
ALTER COLUMN ColumnName datatype;
```

Note-se que a segunda instrução é a que muda de 'ALTER COLUMN' para 'MODIFY COLUMN' ou 'MODIFY', dependendo do SGBD que está a usar. O resto mantém-se inalterável.

Vejamos agora um exemplo. A table em baixo é a “Persons” e contém informação sobre pessoas diferentes.

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Tabela 52 *– Exemplo ‘ALTER TABLE’ (**Fonte:** https://www.w3schools.com/sql/sql_alter.asp)*

Imaginemos que queremos acrescentar a esta tabela uma coluna com o nome “DateofBirth”. Para tal, usaremos a instrução seguinte:

```
ALTER TABLE Persons
ADD DateofBirth date;
```

A nova coluna que acrescentamos à tabela tem como tipo de dados *date*, o que significa que armazena dados em formato de data. Em baixo, é possível ver a tabela com as colunas acrescentadas.

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Tabela 53 *– Exemplo ‘ALTER TABLE’ (**Fonte**: https://www.w3schools.com/sql/sql_alter.asp)*

Contudo, caso mude de ideias e queira alterar o tipo de dados da coluna nova, pode usar a instrução ‘ALTER COLUMN’. Por exemplo, Podemos alterar o tipo de dadosda nova coluna para *year* (ano). Para tal, use a seguinte instrução:

```
ALTER TABLE Persons
ALTER COLUMN DateofBirth year;
```

O tipo de dados *year* apresenta um ano em formato numérico com 2 ou 4 dígitos.

Para apagar a coluna que acabamos de alterar, usamos a instrução ‘DROP COLUMN’.

```
ALTER TABLE Persons
DROP COLUMN DateofBirth;
```

Neste caso, a nossa tabela voltará ao que era no início.

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timotelvn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Tabela 54 *– Exemplo 'ALTER TABLE' (**Fonte**: https://www.w3schools.com/sql/sql_alter.asp)*

'CONSTRAINTS' em SQL

Em SQL, 'Constraints' é usado quando a tabela é criada com a instrução 'CREATE TABLE' ou depois da tabela ter sido criada com a instrução 'ALTER TABLE'.

Síntaxe:

```
CREATE TABLE table_name (
        column1 datatype constraint,
        column2 datatype constraint,
        column3 datatype constraint,
        ....
);
```

O termo 'Constraints' (limitações) é usado para especificar um conjunto de regras e restrições que se aplicam a uma coluna ou tabela. Estas regras e restrições são usadas para assegurar a integridade, rigor e fiabilidade dos dados. Quando aplicado a uma tabela, todas as colunas têm de ceder às limitações impostas.

As seguintes *constraints* são as mais comuns:

- 'NOT NULL';

- ‘UNIQUE’;
- ‘PRIMARY KEY’;
- ‘FOREIGN KEY’;
- ‘CHECK’;
- ‘DEFAULT’;
- ‘CREATE INDEX’.

Iremos agora abordar cada uma destas *constraints* para explicar o seu uso e síntaxe através de exemplos.

‘NOT NULL’ em SQL

Em SQL, as colunas podem ter valores ‘null’ por defeito. A *constraint* ‘NOT NULL’ é usada para evitar valores ‘null’ em colunas. Isto é importante para assegurar que todos os campos necessários são preenchidos quando uma nova entrada é introduzida na tabela.

Imaginemos que queremos criar uma tabela com o nome “Persons” e queremos garantir que as colunas “ID”, “LastName” e “FirstName” não têm qualquer valor ‘null’:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255) NOT NULL,
        Age int
);
```

Se quiser alterar uma tabela ao acrescentar limitações, pode usar a seguinte instrução:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

A limitação ‘UNIQUE’ em SQL

A limitação ‘UNIQUE’ é usada para assegurar que todos os valores de uma coluna não se repetem nas linhas da tabela. Para clarificar, pense na variante ‘ID’. Não é conveniente que duas pessoas tenham a mesma ‘ID’, usamos então a limitação ‘UNIQUE’ para o evitar.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL UNIQUE,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        UNIQUE (ID)
);
```

Como pode ver, dependendo do SGBD que está a usar, há alguns ajustes nos quais a limitação ‘UNIQUE’ é incluída no código.

Caso queira definir uma limitação ‘UNIQUE’ em várias colunas, use o seguinte:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

Também pdoemos acrescentar uma limitação 'UNIQUE' depois de a tabela ter sido criada através da instrução 'ALTER TABLE', que aprendemos anterioremente.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

Para definir uma limitação 'UNIQUE' em várias colunas já existentes, usamos a seguinte instrução:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Persons UNIQUE (ID, LastName);
```

Para eliminar a limitação 'UNIQUE', usamos a seguinte instrução:

MySQL:

```
ALTER TABLE Persons
DROP INDEX UC_Persons;
```

SQL Server/Oracle/ MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Persons;
```

A limitação 'PRIMARY KEY' em SQL

A limitação 'PRIMARY KEY' é usada apenas para identificar cada linha ou registo numa tabela. Note-se que *primary keys* deve conter valores únicos, mas não pode conter valores *null*.

Uma tabela pode ter apenas **UM** *primary key*, que deve ter uma ou várias colunas.

SQL Server/Oracle/MS Access:

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

O exemplo em baixo permite definir uma limitação 'PRIMARY KEY' em várias colunas:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Note-se que a 'PRIMARY KEY' continua a ser apenas uma, mas o valor abrange duas colunas.

Podemos também aplicar uma limitação 'PRIMARY KEY' a uma tabela existente através da seguinte instrução:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

Para acrescentar e definir uma limitação 'PRIMARY KEY', use a seguinte instrução:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Persons PRIMARY KEY (ID, LastName);
```

Para eliminar uma limitação 'PRIMARY KEY', use a seguinte instrução, de acordo com o seu SGBDR:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

A limitação 'FOREIGN KEY' em SQL

A limitação 'FOREIGN KEY' representa as colunas de uma tabela que estão ligadas a uma limitação 'PRIMARY KEY' de outra tabela. À tabela que tem a limitação 'FOREIGN KEY'chama-se chama-se *child table*, e à tabela 'PRIMARY KEY' chama-se *referenced table* ou *parent table*.

Este tipo de limitação é usado para prevenir quaisquer ações que poderiam destruir ligações entre tabelas *child* e *parent*.

Consideremos as seguintes tabelas:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Tabela 55 *– Exemplo de 'FOREIGN KEY' com tabela 'Persons' (**Fonte**: https://www.w3schools.com/sql/sql_foreignkey.asp)*

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Tabela 56 *– Exemplo de 'FOREIGN KEY' com tabela 'Oders' (**Fonte**: https://www.w3schools.com/sql/sql_foreignkey.asp)*

Estas duas tabelas estão ligadas à coluna "PersonID". A *primary key* está na tabela 'Persons', e a *foreign key* corresponde à "PersonID" na tabela 'Orders'.

A limitação 'FOREIGN KEY' previne a inserção de dados inválidos na coluna *foreign key*, porque está ligada à tabela e os seus valores têm de ser idênticos.

Para usar a limitação 'FOREIGN KEY' ao criar uma tabela, podemos usar a seguinte instrução, de acordo com o nosso SGBDR:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
```

```
        OrderNumber int NOT NULL,
        PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

MySQL:

```
CREATE TABLE Orders (
        OrderID int NOT NULL,
        OrderNumber int NOT NULL,
        PersonID int,
        PRIMARY KEY (OrderID),
        FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Esta instrução ligou a tabela “Orders” À tabela “Persons” com a limitação ‘FOREIGN KEY’ tendo por base a coluna “PersonID”.

A limitação ‘CHECK’ em SQL

A limitação ‘CHECK’ é usada para especificar valores numa coluna ou em determinadas colunas de uma tabela tendo por base valores encontrados noutras colunas da mesma linha.

Exemplo de limitação ‘CHECK’ em ‘CREATE TABLE’

O exemplo seguinte é usado par agarantir que uma pessoa não tem menos de 18 anos, para tal é acrescentada a limitação ‘CHECK’ à coluna “Age”:

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
```

```
        CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int CHECK (Age>=18)
);
```

Se quiser nomear uma limitação 'CHECK' e usá-la em várias colunas, pode usar a seguinte instrução:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        City varchar(255),
        CONSTRAINT CHK_Person CHECK (Age>=18 AND City= 'Sandnes')
);
```

Exemplo de limitação 'CHECK' em 'ALTER TABLE'

Para criar uma limitação para uma tabela já existente, use a seguinte instrução:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
To name a constraint and create it on multiple columns, you can use:
ALTER TABLE Persons
```

```
ADD CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’);
```

Exemplo de limitação ‘DROP a CHECK’

Para eliminar uma limitação ‘CEHCK’, pode usar a seguinte instrução de acordo com a o seu SGBDR:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

A limitação ‘DEFAULT’ em SQL

A limitação ‘DEFAULT’ é usada para especificar um valor por defeito para uma coluna. Se não existirem valores especificados, o valor por defeito será acrescentado a todos os novos registos.

Exemplo de limitação ‘DEFAULT’ em ‘CREATE TABLE’

O exemplo seguinte acrescenta um valor por defeito à coluna “City” quando a tabela “Persons” é criada:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
```

Esta limitação também pode ser uasada para inserir valores de Sistema com funções como ‘GETDATE()’:

```
CREATE TABLE Orders (
        ID int NOT NULL,
        OrderNumber int NOT NULL,
        OrderDate date DEFAULT GETDATE()
);
```

Exemplo de limitação ‘DEFAULT’ em ‘ALTER TABLE’

Neste exemplo, a coluna “Cirty” é usada para criar uma limitação ‘DEFAULT’ quando estamos a alterar uma tabela existente:

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT ‘Sandnes’;
```

Oracle:

```
ALTER TABLE Persons
MODIFY City DEFAULT 'Sandnes';
```

Exemplo de limitação 'DROP a DEFAULT'

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

O comando 'CREATE INDEX' em SQL

A instrução 'CREATE INDEX' cria um índice numa tabela. Os índices são úteis quando queremos aceder a dados rapidamente.

Note-se que a atualização de tabelas com índices é mais demorada em comparação com a atualização de tabelas sem índices. Nesta senda, sugere-se a criação de índices apenas em colunas frequentemente consultadas.
Para operar o comando 'CREATE INDEX' numa tabela em que são permitidos valores duplicados, use a seguinte síntaxe:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Para operar o comando 'CREATE UNIQUE INDEX' numa tabela em que não são permitidos valores duplicados, use a seguinte síntaxe:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Lembre-se que a criação de índices varia de base de dados para base de dados, por isso é importante verificar sempre qual a síntaxe a utilizar para criar um na sua base de dados.

Exemplos de 'CREATE INDEX'

Neste exemplo, vamos criar um índice na coluna "LastName" ao especificar o nome "idx_lastname":

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

Para criar um índice numa combinação de colunas, use a seguinte instrução:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

Se quiser, pode acrescentar mais colunas entre os parêntesis.

Exemplos de 'DROP INDEX'

Se quiser apagar um índice, use a seguinte instrução de acordo com o eu SGBDR:

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Auto Increment em SQL

O *Auto increment* é usado para gerar automaticamente números únicos quando um novo registo é introduzido numa tabela. Isto é normalmente usado em *primary key* para assegurar que nenhuma pessoa tem a mesma *ID*.

Este recurso usa diferentes síntaxes em MySQL, SQL Server, Access e Oracle. Em seguida, iremos ver alguns exemplos.

MySQL:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

Em MySQL, 'AUTO_INCREMENT' acrescenta o elemento *auto-increment* por defeito, o valor estabelecido é 1 e este vai aumentando 1 de cada vez.

Se quer que a sequência inicie num valor diferente, use a seguinte instrução:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

Se introduzir um novo registo na tebla "Persons", não terá de especificar o valor para a coluna "PersonsID", pois este será Gerado automaticamente:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

SQL Server

Aqui, estamos a usar o mesmo exemplo que em cima, no qual a coluna “PersonsID” é usada como a p*rimary key na tabela “Persons”:*

```
CREATE TABLE Persons (
        Personid int IDENTITY(1,1) PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

Em SQL Server, o recurso *auto-increment* usa a palavra ‘IDENTIFY’ para ser ativado. Os dois avlores entre parêntesis indicam (o valor inicial e o valor a acrescentar para cada novo registo). Iniciará em 1 e irá aumentando de 1 em 1 a cada valor novo acrescentado.

Por exemplo, se quiser iniciar em 10 e aumentar esse valor de 5 em 5 a cada registo acrescentado, terá de escrever o seguinte ‘IDENTIFY (10,5)’.

Ao acrescentar novos registos não precisa de especificar a “PersonID”, esta será gerada automaticamente, como demonstrado no exemplo em cima.

MS Access

```
CREATE TABLE Persons (
        Personid AUTOINCREMENT PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

O MS Access usa a palavra-chave 'AUTOINCREMENT' para ativar o recurso *auto-increment.* Similarmente aos outros dois, o valor inicial é 1 e a cada novo registo é acrescentado é acrescentado 1.

Pode indicar diferentes valores para valor inicial, como 10, e estabelecer que a cada novo registo o valor vá crescendo de 5 em 5 com "AUTOINCREMENT(10,5)".

Novamente, note que cada vez que acrescentamos um novo registo não é necessário especificar o valor "PersonID", este é gerado automaticamente.

Oracle

Em Oracle, o código é um pouco mais complicado. Para criar um campo *auto-increment*, é necessário criar uma sequência de números.

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

Esta sequência cria um elemento sequência chamado "seq_person", estabelece o valor inicial mínimo (neste caso é 1), e especifíca o aumento por 1. O *cache* especifíca quantos valores de sequência devem ser armazenados na memória para um acesso mais rápido.

Ao contrário dos exemplos anteriores, para introduzir um novo registo na tabela "Persons", é necessário usar a função *nextval.* Esta função é usada para obter o próximo valor do elemento sequência que criámos.

```
INSERT INTO Persons (Personid,FirstName,LastName)
VALUES (seq_person.nextval,'Lars','Monsen');
```

Neste exemplo, podemos ver que a coluna “PersonsID” é selecionada para ser atribuído o próximo número do elemento sequência “seq_person” que criámos.

Datas em SQL

Um dos aspetos mais difíceis ao usar datas é assegurar que o formato que estamos a tentar introduzir é o mesmo da coluna referente a datas na base de dados.

É importante notar que os dados que contêm apenas datas funcionarão como é expectável em consultas. Contudo, se houver também informação referente à hora, as coisas complicam-se um bocado.

Tipos de datas encontrados em MySQL:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Tipos de data encontrados em SQL Server:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Lembre-se que os tipos de data são escolhidos aquando a criação de uma nova tabela na sua base de dados.

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Tabela 57 *– Exemplos de datas com a tabela "Orders" (**Fonte**: https://www.w3schools.com/sql/sql_dates.asp)*

Usaremos a tabela "Orders" no nosso exemplo para selecionar os registos com a "OrderDate" "2088-11-11".

Exemplo:

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11';
```

O resultado esperado assemelhar-se-á a isto:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Tabela 58 *– Exemplo de resultado de pesquisa de "OrderDate" em "Dates" (**Fonte**: https://www.w3schools.com/sql/sql_dates.asp)*

Note-se que duas datas podem ser facilmente comparadas quando não há registo de hora envolvido.

Suponha que tem a tabela “Orders”, mas que a coluna “OrderDate” contém também o registo da hora.

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

Tabela 59 *– Tabela “Orders” com registo de hora em “Dates” (**Fonte**: https://www.w3schools.com/sql/sql_dates.asp)*

Neste exemplo, se tentar usar a mesma pesquisa que usamos em cima não obterá qualquer resultado, pois a pesquisa não considera o registo da hora. É recomendado que o registo da hora não seja usado, a menos que necessário.

```
SELECT *
FROM Orders
WHERE OrderDate='2008-11-11';
```

Views em SQL

Em SQL, uma *view* é uma tabela virtual de um conjunto de resultados criado a partir de uma determinada pesquisa. Uma *view* é útil quando queremos visualizar e apresentar dados através de uma combinação de tabelas.

Síntaxe:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note-se que sempre que o utilizador consulta uma *view, esta* exibe dados atualizados desde que a base de dados recrie a tabela virtual.

Exemplo de como consultar todos os clientes do Brasil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

Para consultar a *view:*

```
SELECT * FROM [Brazil Customers];
```

Outro exemplo é criar uma *view* que selecione todos os produtos da tabela “Products” com o preço superior ao preço médio:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

Para consultar a *view* em cima, use a seguinte instrução:

```
SELECT * FROM [Products Above Average Price];
```

Para atualizar a *view* em cima, use a instrução ‘CREATE OR REPLACE VIEW’:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

O exemplo seguinte adiciona a coluna “City” à *view* “Brazil customers” que criámos anteriormente:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
```

```
WHERE Country = 'Brazil';
```

Para apagar uma *view*, use a instrução 'DROP VIEW':

```
DROP VIEW view_name;
```

Por exemplo, suponha que quer apagar a view "Brazil customers":

```
DROP VIEW [Brazil Customers];
```

4.3. Referências SQL

Palavras-chave SQL

Palavra-chave	Descrição
ADD	Adiciona a coluna a uma tabela existente
ADD CONSTRAINT	Adiciona uma limitação depois de uma tabela ter sido criada
ALL	Apresenta resultados como verdadeiros se os valores de subconsulta corresponderem à condição
ALTER	Acrescenta, apaga ou altera colunas de uma tabela, ou muda o tipo de dados da coluna de uma tabela
ALTER COLUMN	Altera o tipo de dados da coluna de uma tabela

ALTER TABLE	Acrescenta, apaga ou altera as colunas de uma tabela
AND	Apenas inclui linhas em que ambas as condições sejam verdade
ANY	Apresenta resultados como verdadeiros se qualquer um dos valores da subconsulta corresponder à condição
AS	Atribui um nome nove a uma coluna ou tabela com um alias
ASC	Organiza os resultados por ordem crescente
BACKUP DATABASE	Cria uma cópia de segurança de uma base de dados existente
BETWEEN	Seleciona valores dentro de um dado intervalo
CASE	Cria diferentes resultados com base nas condições
CHECK	Uma limitação que restringe o valor a ser introduzido em cada coluna
COLUMN	Altera o tipo de dados de uma coluna ou apaga a coluna de uma tabela

CONSTRAINT	Acrescenta ou elimina uma limitação
CREATE	Cria uma base de dado, um índice, *view*, tabela ou procedimento
CREATE DATABASE	Cria uma nova base de dados SQL
CREATE INDEX	Cria um índice numa tabela (permite valores duplicados)
CREATE OR REPLACE VIEW	Atualiza uma *view*
CREATE TABLE	Cria uma nova tabela numa base de dados
CREATE PROCEDURE	Cria um processo armazenado
CREATE UNIQUE INDEX	Cria um único índice numa tabela (não permite valores duplicados)
CREATE VIEW	Cria uma *view* com base no conjunto de resultados de uma instrução 'SELECT'
DATABASE	Cria ou apaga uma base de dados SQL

DEFAULT	Uma limitação que faculta um valor por defeito para uma coluna
DELETE	Apaga linhas de uma tabela
DESC	Apresenta os resultados por ordem decrescente
DISTINCT	Seleciona apenas valores diferentes
DROP	Apaga uma coluna, limitações, bases de dados, índices, tabelas ou *views*
DROP COLUMN	Apaga a coluna de uma tabela
DROP CONSTRAINT	Apaga uma limitação 'UNIQUE', 'PRIMARY KEY', 'FOREIGN KEY' ou 'CHECK'
DROP DATABASE	Apaga uma base de daos SQL existente
DROP DEFAULT	Apaga uma limitação 'DEFAULT'
DROP INDEX	Apaga o índice de uma tabela
DROP TABLE	Apaga uma tabela de uma base de dados

DROP VIEW	Apaga uma *view*
EXEC	Executa um procedimento armazenado
EXISTS	Testa a existência de registos numa subconsulta
FOREIGN KEY	Uma limitação usada para associar duas tabelas
FROM	Especifica de que tabela selecionar ou apagar dados
FULL OUTER JOIN	Apresenta todas as linhas nas quais existe correspondência entre a tabela da esquerda ou da direita
GROUP BY	Agrupa os resultados (usada com as funções combinadas: 'COUNT', 'MAX', 'MIN', 'SUM', 'AVG')
HAVING	Usado em vez de 'WHERE' com funções combinadas
IN	Permite especificar vários valores numa instrução 'WHERE'
INDEX	Cria ou apaga um índice numa tabela
INNER JOIN	Apresenta linhas que têm valores correspondentes em ambas as tabelas

INSERT INTO	Insere novas linhas numa tabela
INSERT INTO SELECT	Copia dados de uma tabela para outra
IS NULL	Testa valores vazios (*null)*
IS NOT NULL	Testa valores não vazios (not *null)*
JOIN	Junta tabelas
LEFT JOIN	Exibe todas as linhas da tabela da esquerda e as linhas correspondentes da tabela da direita
LIKE	Procura um padrão específico numa coluna
LIMIT	Determina o número de registos a apresentar nos resultados
NOT	Inclui apenas linhas nasquais a condição não seja verdade
NOT NULL	Uma limitação que impõe que uma coluna não aceite valores *null*
OR	Inclui linhas em que uma ou outra condição sejam verdade

ORDER BY	Organiza os resultados por ordem crescente ou decrescente
OUTER JOIN	Exibe todas as linhas quando existe correspondência na tabela da esquerda ou da direita
PRIMARY KEY	Uma limitação que apenas identifica cada registo numa tabela da base de dados
PROCEDURE	Um procedimento armazenado
RIGHT JOIN	Apresenta todas as linhas da tabela da direita e as linhas correspondentes da tabela da esquerda
ROWNUM	Especifica o número de registos a apresentar nos resultados
SELECT	Seleciona dados de uma base de dados
SELECT DISTINCT	Seleciona apenas valores diferentes
SELECT INTO	Copia dados de uma tabela para um atabela nova
SELECT TOP	Determina o número de registos a apresentar nos resultados

SET	Determina que colunas e valores devem ser atualizados numa tabela
TABLE	Cria uma tabela ou adiciona, apaga ou altera colunas de uma tabelaCreates a table, or adds, deletes, or modifies columns in a table, ou apaga uma tabela ou os dados de uma tabela
TOP	Determina o número de registos a apresentar nos resultados
TRUNCATE TABLE	Apaga os dados de uma tabela, mas não a tabela
UNION	Combina os resultados de duas ou mais instruções 'SELECT' (apenas valores diferentes)
UNION ALL	Combina os resultados de duas ou mais instruções 'SELECT' (permite valores duplicados)
UNIQUE	Uma limitação que garante que todos os valores de uma coluna são únicos
UPDATE	Atualiza as linhas de uma tabela
VALUES	Determina os valores de uma instrução 'INSERT INTO'

VIEW	Cria, atualiza ou apaga uma *view*

Tabela 68 *– Palavras-chave SQL e respetivas descrições (**Fonte:** https://www.w3schools.com/sql/sql_ref_keywords.asp)*

Funções em MySQL

Para mais informações sobre funções usadas especificamente em MySQL, os aprendentes podem consultar o seguinte link.

Funções em SQL Server

Para mais informações sobre funções usadas especificamente em SQL Server, os aprendentes podem consultar o seguinte link

Funções em MS Access

Para mais informações sobre funções usadas especificamente em MS Access, os aprendentes podem consultar o seguinte link.

SQL Quick Ref

Para saber mais sobre instruções usadas em SQL e as suas síntaxes, os aprendentes podem consultar este link.

Regra geral, todas as colunas de uma tabela requerem um nome e tipo de dados.

Um programador SQL terá de decidir que tipo de dados irão ser armazenados em cada coluna quando criar a tabela. O tipo de dados é usado para a SQL compreender os dados que cada coluna irá conter e como esta nirá interagir com os dados.

Tenha com consideração que o tipo de dados pode ter nomes diferentes em bases de dados diferentes. Confirme sempre os documentos, mesmo que o nome seja o mesmo, pois os detalhes podem diferir em alguns aspetos, como, por exemplo, em tamanho.

Tipos de dados em MySQL (Versão 8.0)

O MySQL tem três tipos de dados: *string*, numérico e data/hora.

Tipos de dados *String*

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Tabela 60 *– Tipos de dados 'String' (MySQL) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Numéricos

Data type	Description
BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
TINYINT(size)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(size)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)
MEDIUMINT(size)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
INTEGER(size)	Equal to INT(size)
BIGINT(size)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)
FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
DOUBLE PRECISION(size, d)	
DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.
DEC(size, d)	Equal to DECIMAL(size,d)

Tabela 61 *– Tipos de dados Numéricos (MySQL) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Data/Hora

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(fsp)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(fsp)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(fsp)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Tabela 62 *– Tipos de dados Data/Hora (MySQL) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados em SQL Server

Tipos de dados *String*

Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Tabela 63 *– Tipos de dados 'String' (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Numéricos

Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 -1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 -1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

*Tabela 64 – Tipos de dados Numéricos (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados Data/Hora

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

*Tabela 65 – Tipos de dados Data/Hora (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Outros tipos de dados

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

*Tabela 66 – Outros tipos de dados (SQL Server) (**Fonte**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Tipos de dados em MS Access

Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large Objects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

Tabela 67 *– Tipos de dados em Access (**Fonte:** https://www.w3schools.com/sql/sql_datatypes.asp)*

4.4. Exemplos SQL

Exemplos SQL

Existe uma vasta lista de exemplos no *website* W3Schools que os aprendentes podem consultar para estudo individual e praticar as suas competências em SQL.

Questionário SQL

Para os aprendentes que quiserem avaliar o seu conhecimento e competeências em SQL, por favor indique um dos seguintes *websites:*

- W3Schools
- Tutorialspoint