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

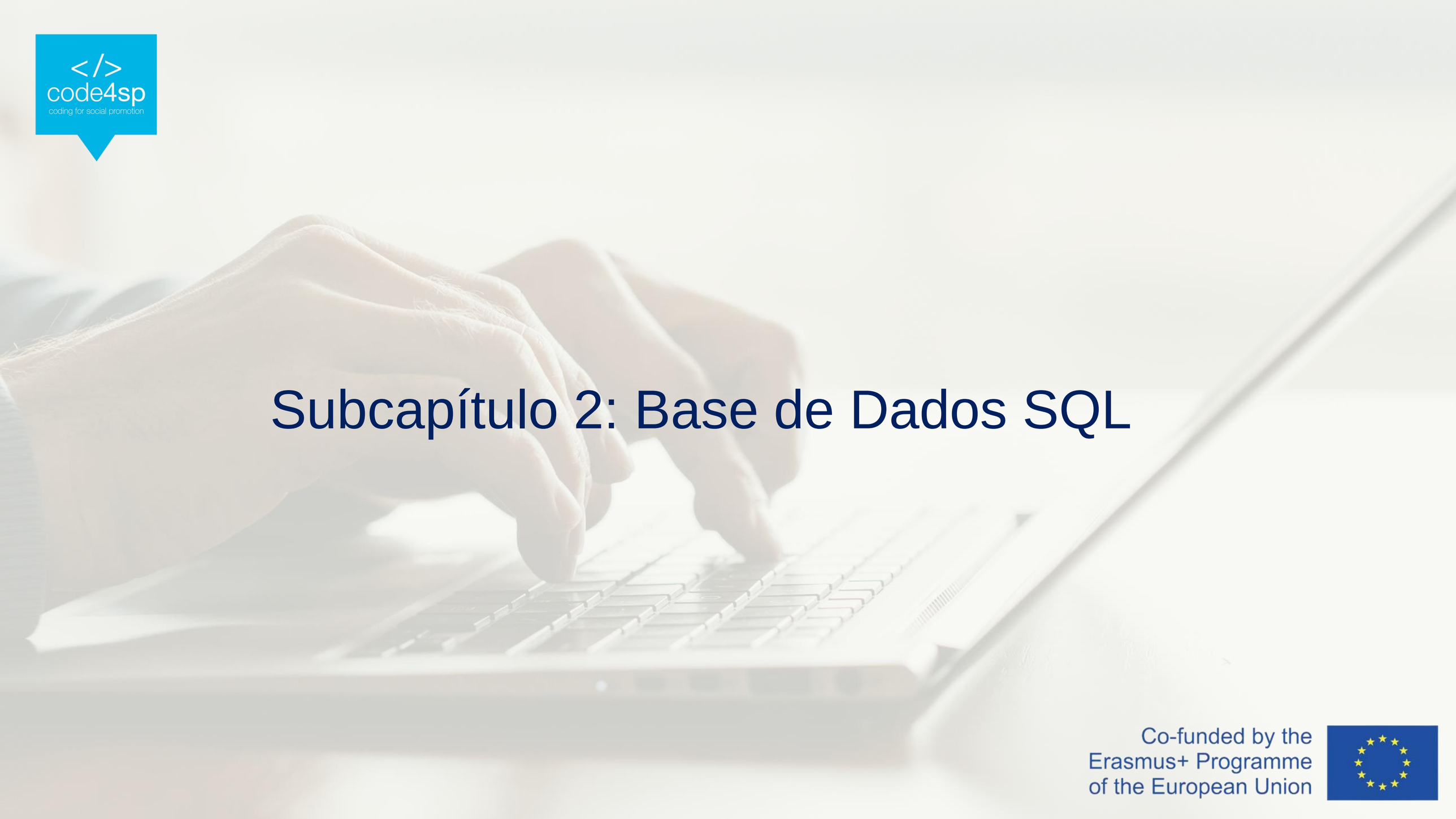
code4sp
coding for social promotion

Materiais de Formação de SQL
Subcapítulo 2 – Base de Dados SQL

WP3: Materiais de Formação Code4SP

Preparado por: CITIZENS IN POWER

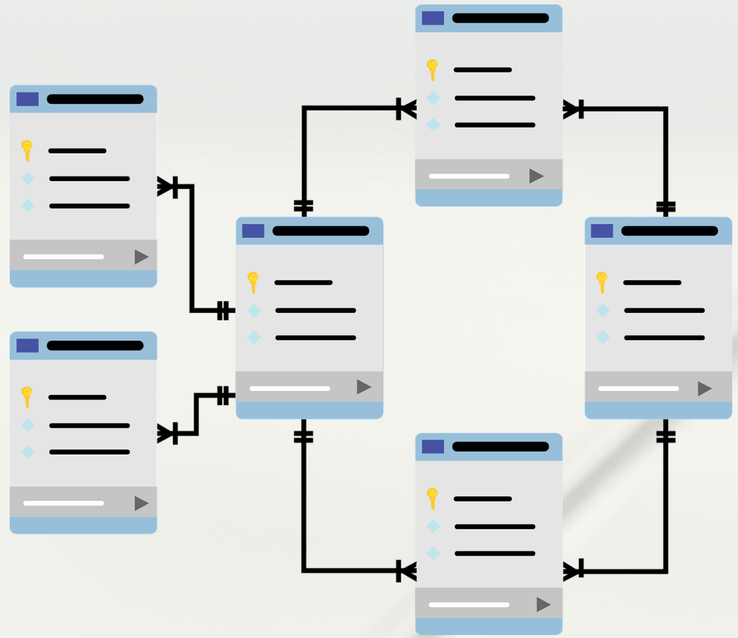
The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the informatoion contained therein. (Project Number: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN)



Subcapítulo 2: Base de Dados SQL

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

Introdução às 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 progressivamente para instruções mais complicadas.

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

‘CREATE DATABASE’ em SQL

A instrução ‘CREATE DATABASE’ cria uma **nova base de dados SQL.**

Síntaxe:

CREATE DATABASE DatabaseName;

Note: É importante recordar que o nome da base de dados deve ser único no Sistema de Gestão de Base de dados Relacional que está a usar, e garantir 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;

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

‘DROP DB’ em SQL

A instrução ‘DROP DATABASE’ **elimina as bases de dados SQL existentes.**

Síntaxe:

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.

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

Exemplo:

DROP DATABASE testDB;

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

‘BACKUP DB’ em SQL

A instrução ‘BACKUP DATABASE’ faz uma cópia de segurança completo de uma base de dados SQL existente.

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

Síntaxe:

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

Exemplo:

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

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

‘BACKUP DB’ em SQL

- Para evitar problemas técnicos, é melhor guardar a cópia de segurança numa unidade de disco **diferente daquela 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 cópia de segurança feita. Este tipo de cópia de segurança reduz o tempo desta operação.

Síntaxe:

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

Exemplo:

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

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

‘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.

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

‘CREATE TABLE’ em SQL

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)
);
```

PersonID	LastName	FirstName	Address	City

Tabela vazia – Exemplo CREATE TABLE

(Fonte: https://www.w3schools.com/sql/sql_create_table.asp)

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

‘CREATE TABLE’ em SQL

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

Síntaxe:

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

Exemplo:

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

Tal como aprendemos na secção 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.

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

‘DROP TABLE’ 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;
```

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

‘DROP TABLE’ em SQL

Também pode escolher apagar apenas os dados da tabela 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;

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

‘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.

Síntaxe para adicionar 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);
```

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

‘ALTER TABLE’ em SQL

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

É importante lembrar 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;
```

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

‘ALTER TABLE’ em SQL

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

- ALTER COLUMN (for SQL Server/MS Access);
- MODIFY COLUMN (for My SQL/ Oracle prior to version 10G);
- MODIFY (for Oracle version 10G and later).

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.

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

‘ALTER TABLE’ em SQL

Exemplo: Acrescentar uma coluna denominada “DateofBirth” na tabela Persons

```
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 formatoo 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	

Exemplo ALTER TABLE

(Fonte: https://www.w3schools.com/sql/sql_alter.asp)

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

‘ALTER TABLE’ em SQL

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):

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

O tipo de dados year apresenta um ano em formatoo 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;
```

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

‘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’.

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.

Síntaxe:

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

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

‘CONSTRAINTS’ em SQL

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.

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

‘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
);
```

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

‘NOT NULL’ em SQL

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

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

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

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
);
```

My SQL:

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

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

A limitação ‘UNIQUE’ em SQL

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)
);
```

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

A limitação ‘UNIQUE’ em SQL

Também podemos acrescentar uma limitação ‘UNIQUE’ depois de a tabela ter sido criada através da instrução ‘ALTER TABLE’, que aprendemos anteriormente.

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);
```

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

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)
);
```

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

A limitação ‘PRIMARY KEY’ em SQL

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.**

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

A limitação ‘PRIMARY KEY’ em SQL

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);
```

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

A limitação ‘PRIMARY KEY’ em SQL

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;
```

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

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. **Considerando as tabelas abaixo, o que terão elas em comum?**

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

Tabela Personel - Exemplo FOREING KEY (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 Orders - Exemplo FOREING KEY (Fonte: https://www.w3schools.com/sql/sql_foreignkey.asp)

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

A limitação ‘FOREIGN KEY’ em SQL

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.

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)
);
```

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

A limitação ‘FOREIGN KEY’ em SQL

My SQL:

```
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”.

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

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 da limitação CHECK em CREATE TABLE: O exemplo seguinte é usado para garantir 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)
);
```

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

A limitação ‘CHECK’ em SQL

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’)
);
```

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

A limitação ‘CHECK’ em SQL

Exemplo da limitação CHECK na 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’);
```

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

A limitação ‘CHECK’ em SQL

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;
```

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

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'
);
```

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

A limitação ‘DEFAULT’ em SQL

Esta limitação também pode ser usada para inserir **valores de sistema com funções como ‘GETDATE()’**

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

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

A limitação ‘DEFAULT’ em SQL

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';
```

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

A limitação ‘DEFAULT’ em SQL

Exemplo de limitação ‘DROP a DEFAULT’

É usado para eliminar a limitação por defeito da tabela já existente

MySQL:

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

SQL Server / Oracle / MS Access:

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

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

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.

Para operar o comando ‘CREATE INDEX’ numa tabela em que são permitidos valores duplicados, usa-se 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, ...);
```

- ✓ Note-se que tabelas com índices levam mais tempo a atualizar em comparação com tabelas sem. Por conseguinte, sugere-se a **criação de índices apenas em colunas que são frequentemente pesquisadas.**
- ✓ Tenha em mente que a criação de índices varia de base de dados para base de dados, por isso verifique sempre a síntaxe para criar um na sua base de dados.

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

O comando ‘CREATE INDEX’ em SQL

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.

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

O comando ‘CREATE INDEX’ em SQL

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;

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

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.

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

Auto Increment em SQL

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 for desejável que a sequência inicie num valor diferente, usar a seguinte instrução:

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

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

Auto Increment em SQL

Se introduzir um novo registo na tabela “Persons”, não terá de especificar o valor para a coluna “PersonsID”, pois este será gerado automaticamente:

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

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

Auto Increment em SQL

SQL Server

Aqui, estamos a usar o mesmo exemplo que em cima, no qual a coluna “PersonsID” é usada como a primary 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 anterior.

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

Auto Increment em SQL

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

.

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

Auto Increment em SQL

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.

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

Auto Increment em SQL

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.

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

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 - formatoo AAAA-MM-DD
- DATETIME - formatoo: AAAA-MM-DD HH:MI:SS
- TIMESTAMP - formatoo: AAAA-MM-DD HH:MI:SS
- YEAR - formatoo AAAA ou AA

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

Tipos de datas encontrados em MySQL:

Tipos de data encontrados em SQL Server::

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

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

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

Exemplo:

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

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

Datas em SQL

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

Exemplo de resultado de pesquisa de “OrderDate” 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.**

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

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.**

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

Views em SQL

Exemplo de como 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];
```

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];
```

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

Views em SQL

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;
```

Para apagar a view "Brazil customers":

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

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

Tipos de dados SQL

Geralmente, cada coluna de uma tabela requer um nome e um tipo de dados. Um programador SQL terá de decidir o tipo de dados que serão armazenados dentro de cada coluna ao criar uma tabela.

O tipo de dados é utilizado para SQL para compreender os dados que serão contidos em cada coluna e também como irá interagir com os dados.

*** Tenha em mente que os tipos de dados podem ter nomes diferentes em bases de dados diferentes. Verifique sempre a documentação, mesmo que o nome seja o mesmo, porque outros detalhes podem ser diferentes, como o tamanho.**

Para mais informações sobre diferentes tipos de dados em diferentes RDBMS, visite o seguinte website: https://www.w3schools.com/sql/sql_datatypes.asp

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

Vamos praticar?

Já aprendemos muita coisa, é hora de praticar!

Para isso, cliquemos aqui.

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

OBRIGADO!

PRÓXIMO CAPÍTULO: Referências SQL

Para mais informação, clicar aqui

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