

Upgrading Skills

Primavera
ACADEMY

Manual de Formação **T-SQL**

2019-v0.1-GB

ÍNDICE

INTRODUÇÃO AO T-SQL	4
PLANO DO CAPÍTULO	4
OBJETIVOS.....	4
O MOTOR DA BASE DE DADOS.....	5
A INTERAÇÃO COM O MOTOR DE BASE DE DADOS	6
PRINCIPAIS CARACTERÍSTICAS	6
COMO ESCREVER COMANDOS T-SQL	7
EXERCÍCIOS (BLOCO 1):	14
EXERCÍCIOS (BLOCO 2):	19
FUNÇÕES ADICIONAIS	20
FUNÇÕES ADICIONAIS - CADEIAS DE CARACTERES	24
FUNÇÕES ADICIONAIS – AGREGAÇÃO	26
EXERCÍCIOS (BLOCO 3):	28
JUNÇÃO DE TABELAS (JOIN)	29
EXERCÍCIOS (BLOCO 4):	33
EXERCÍCIOS (BLOCO 5):	42
MANIPULAÇÃO DE DADOS.....	43
EXERCÍCIOS (BLOCO 6):	46
ANEXO – EXERCÍCIOS.....	47
OBJETIVOS.....	47
EXERCÍCIOS DE T-SQL	47

As informações descritas neste documento podem ser alteradas sem aviso prévio. Todos os nomes apresentados ao longo dos exemplos e imagens, são fictícios. Nenhuma parte deste documento pode ser reproduzida ou transmitida sob qualquer formato, para qualquer propósito, sem autorização escrita da PRIMAVERA BSS. Excetua-se a transcrição de certas passagens para efeitos de apresentação, crítica ou discussão das ideias e opiniões contidas no livro. Esta exceção não pode, porém, ser interpretada como permitindo a transcrição de textos em recolhas antológicas ou similares, da qual pode resultar prejuízo para o interesse para a obra. Os infratores são passíveis de procedimento judicial.

Introdução ao T-SQL

Plano do capítulo

Breve abordagem histórica da Linguagem T-SQL

As instruções T-SQL (DDL, DML e DCL)

Objetivos

Depois deste capítulo deverá ser capaz de:

- Entender o tipo de operações que se podem executar via T-SQL
- Executar operações e consulta de dados (SELECT)
- Efetuar ligações entre tabelas (JOIN)
- Executar operações de inserção (INSERT), atualização, (UPDATE) e eliminação (DELETE)

Acerca do SQL

O SQL (Structured Query Language) é uma Linguagem Estruturada de Consulta, baseada na língua Inglesa, que permite definir pedidos de informação/ comandos, que são emitidos ao motor da base de dados, e por este processados.

Segundo a **ANSI** (American National Standards Institute), o SQL é a linguagem standard para bases de dados relacionais. Alguns exemplos de Sistemas de Gestão de Bases de Dados Relacionais (SGBDR) que usam o SQL são o Microsoft SQL Server, Oracle, Sybase, DB2, Access e muitos outros. Apesar do standard ANSI-SQL ser um facto, alguns SGBDR implementam extensões ao SQL, que, na maior parte dos casos, só podem ser usadas no próprio sistema. No entanto, o conjunto base de cláusulas SQL (SELECT, INSERT, UPDATE, DELETE, CREATE e DROP) podem ser utilizados para levar a cabo a maior parte das operações necessárias.

Em termos lógicos, podemos dividir o SQL em três subconjuntos diferentes:

- **DML** - Data Manipulation Language (Manipulação de dados): Subconjunto que inclui as cláusulas, ou comandos, SQL que permitem usar e manipular os dados: SELECT, UPDATE, DELETE e INSERT.

SELECT – consulta de dados

UPDATE – atualização/alteração de registos

DELETE – apaga registos

INSERT – insere novos registos

Neste curso, vamo-nos focar essencialmente nestes comandos.

- **DDL** - Data Definition Language (Definição de estrutura da base de dados): Cláusulas que permitem modelar a estrutura da base de dados (criar, modificar e remover objetos ou a própria base de dados). Exemplos: CREATE TABLE, ALTER TABLE, DROP TABLE

CREATE TABLE – cria tabela

ALTER TABLE – modifica tabela

DROP TABLE – apaga tabela

- **DCL** - Data Control Language (Controlo de permissões): Cláusulas que permitem controlar permissões nos objetos da base de dados. Exemplos: GRANT, REVOKE.

GRANT – concede privilégios de acesso a utilizadores

REVOKE - remove privilégios de acesso a utilizadores

Este capítulo tem por objetivo fornecer-lhe conhecimentos base de T-SQL.

Ao nível das aplicações, não terá necessidade de alterar a estrutura da base de dados, que é definida previamente, bem como controlar acessos aos dados. Assim, vamos centrar este capítulo no estudo passo a passo da linguagem SQL em geral, e da cláusula que lhe será mais útil: o SELECT, juntamente com a descrição de uma grande variedade de opções, cláusulas e métodos a ele associados.

A explicação exaustiva dos restantes temas UPDATE, DELETE, INSERT, DDL e DCL não é incluída, para não tornar este documento demasiado extenso e de leitura menos acessível.

O motor da base de dados

O motor da base de dados¹ é responsável pelas seguintes tarefas:

- Gestão e armazenamento dos dados, para além da sua definição
- Controlo de acesso aos dados (restrições, acessos simultâneos, etc.)
- Interpretação da linguagem SQL
- Controlo de integridade dos dados, mecanismos anticorrupção e segurança

O motor da base de dados é assim a única entidade que tem acesso direto aos dados.

¹ do inglês: DATABASE ENGINE

A interação com o motor de base de dados

Para que seja possível aceder aos dados, é assim necessário comunicar ao motor de base de dados que informação é pretendida ou que operações desejamos executar sobre os dados, para que este aceda a eles diretamente, e retorne os resultados.

Esta arquitetura pode ser visualizada na imagem seguinte:

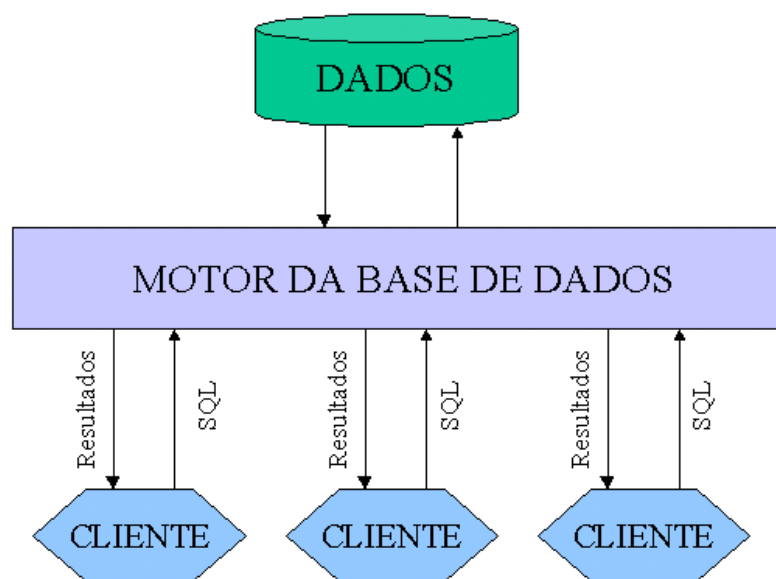


Fig. 1 Arquitetura

A linguagem para troca de pedidos/comandos é, na esmagadora maioria dos sistemas de gestão de bases de dados, o T-SQL.

Principais características

O SQL esteve, desde a sua origem, associado a bases de dados relacionais.

O facto do SQL ser baseado na língua Inglesa é particularmente evidente se atentarmos nos seus comandos (cláusulas): SELECT para seleccionar, INSERT para Inserir, CREATE para criar, DELETE para apagar, FROM para designar origem, WHERE para designar condições, etc.

O SQL é, na sua essência, uma Linguagem Declarativa: em SQL indicamos a informação que pretendemos obter, e não a forma como esta deve ser obtida. O motor da base de dados encarregar-se-á de escolher os métodos mais adequados para responder aos comandos/pedidos T-SQL, e retornar os resultados.

Este facto contrasta com as linguagens de programação usuais, em que devem ser especificadas instruções que descrevem ao sistema como deve executar as ações, por forma a conseguir os resultados pretendidos. Estas são Linguagens Procedimentais.

As linguagens declarativas são por isso bastante mais simples de usar.

Como escrever comandos T-SQL

As Regras

As regras para escrita de comandos T-SQL são bastante simples:

- Um comando T-SQL pode ocupar mais do que uma linha;
- Os comandos/cláusulas são geralmente colocados em linhas separadas;
- Não há distinção entre maiúsculas e minúsculas;
- As cláusulas SQL e restantes elementos devem estar separados por espaços.
- Carateres espaço e tabs adicionais são ignorados.

Exemplo:

SELECT * equivale a **Select * From Clientes**
FROM Clientes

Um comando SQL é habitualmente designado por Query ou Consulta.

Seleção de dados (SELECT)

A cláusula/comando SELECT permite solicitar ao motor de base de dados um conjunto de dados, de acordo com um determinado critério.

A sintaxe, na sua forma mais simples, é a seguinte:

```
SELECT Coluna1 [, Coluna2 ....]  
FROM Tabela1 [, Tabela2 ...]  
[WHERE Condição1, [, Condição2 ...] ]
```

Vejamos um exemplo muito simples:

```
SELECT Nome
FROM Clientes
WHERE CodTerc = 1
```

- (retorna "Paulo", quando aplicado à tabela exemplo da figura)

No fundo, especificamos:

- **Que dados obter:** Especificamos, a seguir ao SELECT, o nome da(s) coluna(s) (campos) desejada(s). Neste caso, queremos a coluna Nome.
- **De onde os obter:** Uma base de dados contém um grande número de tabelas. Adicionalmente, os nomes de campos podem estar repetidos entre tabelas. Assim, existe a necessidade de especificar a que tabela os campos dizem respeito.

No exemplo corrente, a tabela é Clientes, e especificámo-la a seguir ao **FROM**.

- **Que condições devem satisfazer (opcional):** Se não especificarmos qualquer condição, serão retornados todos os registos da tabela. Se desejarmos obter apenas os registos que satisfazem determinado critério, especificamo-lo na área de condições. Neste caso, queremos o Nome do cliente cujo código é 1. Para o efeito, adicionamos a condição "CodTerc = 1" a seguir à cláusula WHERE.

O asterisco (*) pode ser usado para especificar todas as colunas duma tabela.

Mais um exemplo, para seleccionar todos os registos da tabela de Clientes, usaríamos:

```
SELECT *
```

```
FROM Clientes
```

Para seleccionar todas as colunas do cliente com o código 1, ou seja, todos os dados do cliente com o código 1, usaríamos:

```
SELECT *
```

```
FROM Clientes
```

```
WHERE CodTerc = 1
```

Condições Compostas

Existem algumas cláusulas específicas que poderá usar nas suas condições.

E lógico - AND

Junta duas ou mais condições, e mostra resultados apenas se todas as condições forem satisfeitas.

```
SELECT Nome
```

```
FROM Clientes
```


WHERE CodPostal = 4700 AND Vendedor = '1'

- Só serão selecionados os Clientes de Braga (4700) do Vendedor 1.

OU lógico - OR

Junta duas ou mais condições, e mostra resultados se qualquer das condições for satisfeita.

SELECT Nome

FROM Clientes

WHERE CodPostal = 4700 OR CodPostal = 4000

- Só serão selecionados os Clientes de Braga (4700) e Porto (4000)

Negação - NOT

Pode usar a negação em conjunto com outros operadores. Por exemplo, tendo como base a query apresentada na secção anterior, que selecionava todos os Clientes de Braga e do Porto, é muito fácil obter uma versão oposta: todos os Clientes que não moram nem em Braga nem no Porto:

SELECT Nome

FROM Clientes


WHERE NOT (CodPostal = 4700 OR CodPostal = 4000)

Condições e Operadores

Existe um leque familiar de operadores que pode usar nas condições:

Operador	Significado	Exemplo
=	Igual a	=10 ='Braga'
>	Maior que	>1000 >'a'

>=	Maior ou igual a	>= 10 >= 'Ant'
<	Menor que	<100 <'Man'
<=	Menor ou igual a	<=100 <='Man'
<>	Diferente de	<>10 <>'Braga'
LIKE	Parecido com	LIKE 'Ant%'
IS NULL	Valores nulos	Localidade is null

 O operador % é aplicável apenas no âmbito do operador "LIKE"

Intervalos - BETWEEN

Permite especificar uma gama de valores que deve ser respeitada por um campo. Exemplo: todos os Artigos com preços de venda entre 1000 e 5000:

```
SELECT CodArtigo, Designacao, PrecoVenda
FROM Artigos
WHERE PrecoVenda BETWEEN 1000 AND 5000
```

Comparando com listas de valores – IN

O operador IN permite verificar se um determinado valor corresponde a qualquer dos valores numa lista ou SubQuery. Na sua utilização mais simples, o IN permite verificar a correspondência entre valores e uma lista. Suponha que deseja visualizar todos os Clientes que vivem em Braga, Porto e Lisboa. Pode fazê-lo inspecionando o campo Localidade e impondo uma condição OR:

```
SELECT Nome, Localidade
FROM Clientes
WHERE Localidade = 'Braga'
```

OR Localidade = 'Porto'

OR Localidade = 'Lisboa'

Usando o IN, poderá simplificar o seu comando T-SQL, especificando a lista de valores admitidos:

```
SELECT Nome, Localidade  
FROM Clientes  
WHERE Localidade IN ('Braga', 'Porto', 'Lisboa')
```

Pode usar a cláusula IN em conjunto com a negação, para obter outros resultados. Por exemplo, todos os Clientes que não vivem em nenhuma das três cidades:

```
SELECT Nome, Localidade  
FROM Clientes  
WHERE Localidade NOT IN ('Braga', 'Porto', 'Lisboa')
```

Pseudónimos de Colunas e Tabelas

É possível atribuir novos nomes aos campos retornados por um SELECT. Supondo que deseja usar o campo CodTerc de Faturas como sendo CodCliente:

```
SELECT CodTerc AS CodCliente, Nome, Num, Data  
FROM Fact
```

De igual modo, é possível usar em comandos T-SQL mais complexos abreviaturas para os nomes das tabelas. Este facto é particularmente útil nos casos em que estamos a seleccionar informação proveniente de várias tabelas.

```
SELECT c.Nome, c.CodTerc, f.Num, f.Data  
FROM Clientes c, Faturas f
```

Eliminando Duplicados - DISTINCT

Esta cláusula, quando incluída no seu comando T-SQL, vai fazer com que cada linha apareça uma só vez, ou seja, não existam duplicados.

Suponha que deseja listar todas as localidades nas quais a sua empresa tem Clientes. Como já mencionámos, o campo Localidade faz parte da tabela de Clientes.

Como é óbvio, podem existir vários Clientes por localidade. Se o seu comando for:

```
SELECT Localidade  
FROM Clientes
```

Provavelmente, existirão localidades que aparecem mais do que uma vez (uma por cada Cliente dessa Localidade).

Como pretendemos que cada localidade apareça uma só vez, adicionamos DISTINCT à nossa query:

```
SELECT DISTINCT Localidade  
FROM Clientes
```

A Ordenação dos dados - ORDER BY

Por vezes sentirá a necessidade de impor ordenações específicas aos seus dados. Por omissão, é usada a ordenação imposta pelo índice da tabela, definido na base de dados. No entanto, usando a cláusula ORDER BY, pode apresentar os dados ordenados por qualquer campo, ou conjunto de campos.

Por exemplo, ao seleccionar o código dum Cliente, o seu Nome e a Localidade onde reside, terá várias hipóteses de ordenação:

Por Nome :

```
SELECT CodTerc, Nome, Localidade  
FROM Clientes  
ORDER BY Nome
```

Por Código:

```
SELECT CodTerc, Nome, Localidade
```

FROM Clientes

ORDER BY CodTerc

Por Localidade:

SELECT CodTerc, Nome, Localidade

FROM Clientes

ORDER BY Localidade

Adicionalmente, pode especificar a ordenação por mais do que um campo. Nesse caso, separe os campos por vírgulas na cláusula ORDER BY.

É possível especificar o tipo de ordenação desejado. Por omissão é assumida a ordenação ascendente, o que equivalente à utilização da cláusula ASC. Para forçar a ordenação descendente, é usada a cláusula DESC:

SELECT CodTerc, Nome, Localidade

FROM Clientes

ORDER BY Nome ASC

SELECT CodTerc, Nome, Localidade

FROM Clientes

ORDER BY Nome DESC

Exercícios (bloco 1):

1. Listar todos os clientes existentes.
2. Listar todos os clientes existentes, apresentando apenas o seu código e nome.
3. Listar artigos em que o stock mínimo está definido acima de 10.
4. Listar clientes com limite de crédito superior a 200.000 ordenados por nome do Cliente de forma descendente.
5. Listar clientes com limite de crédito entre 150.000 e 300.000.
6. Listar clientes com limite de crédito diferente de 150.000.
7. Listar clientes cuja localidade é Porto.
8. Listar clientes cuja localidade é Lisboa ou qualquer outra começada por B.
9. Listar clientes cuja localidade não é Lisboa.
10. Listar clientes cuja localidade é Porto, Lisboa ou Braga.
11. Listar clientes em que o país não está preenchido.
12. Listar quais os artigos que já foram registados em documentos de venda.

Criando Campos Calculados

É muito simples criar campos calculados baseados em campos reais da base de dados.

Supondo que temos a tabela seguinte (Lfact):

Num	TipoDoc	CodArtigo	PrecoUnit	Qtd
1	FA	A0001	125.00	5
2	FA	B0004	133.20	3

Nesta tabela simples (versão simplificada dos detalhes duma fatura), temos um código que identifica um artigo, o seu preço unitário e uma quantidade.

Operador	Significado	Exemplo
*	Multiplicação	SELECT PrecUnit * Qtd AS PrecoFinal FROM Lfact
+	Soma	SELECT PrecUnit + 10 AS PrecoAcrescido FROM Lfact
-	Subtracção	SELECT PrecUnit – 50 AS PrecoDescontado FROM Lfact
/	Divisão	SELECT Percentagem / 100 AS FactPerc FROM Teste
MAX	Máximo	SELECT MAX(Num)+ 1 As ProxFatura FROM LFact
MIN	Mínimo	SELECT MIN(PrecUnit) AS PrecoMaisBaixo FROM Lfact
AVG	Média Aritmética	SELECT AVG(PrecUnit) AS PrecoMedio FROM Lfact
COUNT	Contagem	SELECT COUNT(CodArtigo) AS NumArtigos FROM Lfact
SUM	Somatório	SELECT SUM(Qtd) AS Quantidade FROM Lfact

Agrupamentos - GROUP BY

A Cláusula GROUP BY permite criar sumários da informação através do T-SQL. Quando estabelecemos agrupamentos dentro de uma tabela, estamos (conceptualmente) a dividir a tabela em partições distintas, de acordo com a variação de um determinado atributo (campo) ou vários.

Suponhamos agora que desejamos saber o número de faturas pendentes de cada cliente.

A função de agregação a usar é o COUNT.

Um comando do tipo

```
SELECT Entidade, COUNT(Tipodoc) AS NumFaturas  
FROM Pendentes  
WHERE tipodoc = 'fa'
```

não é válido. O SQL Server retorna uma mensagem do tipo:

Column 'Pendientes.Entidade' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

Ou seja, não podemos incluir o campo Entidade a menos que:

- Esteja incluído numa função de agregação (o que não pretendemos)
- Esteja definido um grupo.

Um agrupamento é definido usando a cláusula GROUP BY, e adicionando o nome do campo segundo o qual desejamos agrupar.

A melhor forma de contar o n.º de faturas pendentes de cada Cliente é particionar os dados por cada Cliente, e aplicar a função COUNT a cada partição. Ou seja, definir um agrupamento por cliente. Para o efeito, acrescentamos a cláusula GROUP BY Entidade ao nosso comando T-SQL.

Na realidade, o SQL Server exige que inclua na lista dos campos de agrupamento todos os campos incluídos no SELECT que não estejam a ser usados numa função de agregação. No caso presente, estamos a seleccionar um campo: Entidade. O procedimento comum é incluir primeiro o campo (ou campos) segundo o qual queremos agrupar (Entidade), e depois os restantes (caso existam).

Assim, o nosso comando passa a ser:

```
SELECT Entidade, COUNT(Tipodoc) AS NumFaturas  
FROM Pendentes  
WHERE tipodoc = 'fa'  
GROUP BY Entidade
```


Vejamos outros exemplos possíveis:

- Número de Clientes por Localidade:

```
SELECT Localidade, Count(CodTerc) AS 'Numero de Clientes'  
FROM Clientes  
GROUP BY Localidade
```

Resultado:

Localidade	Numero de Clientes

NULL	21
Adaufe	15
Alfragide	5
Amadora	10
Amares	7
Barcelos	59
Braga	285
Cabeceiras de Bastos	5
Caldas das Taipas	200
Carcavelos	20
Castelo de Paiva	12
...	

- Preços Mínimo, Máximo e Médio de venda de cada Artigo, com base nas faturas:

```
SELECT CodArtigo,  
       MIN(PrecoUni) AS PrecoMinimo,  
       MAX(PrecoUni) AS PrecoMaximo,  
       ROUND(AVG(PrecoUni),2) AS PrecoMedio  
FROM LFact  
GROUP BY CodArtigo
```

Resultado:

CodArtigo	PrecoMinimo	PrecoMaximo	PrecoMedio
0101001PII333INTEL	1343.0	375554.0	52713.053
0101001PII350AMD	2000.0	74737.0	38368.5
0101001PII400BOLT	1000.0	1000.0	1000.0
0101001UPGRADE	20051.0	20051.0	20051.0
...			

Impondo condições aos grupos - HAVING

O HAVING está para os grupos assim como o WHERE está para as linhas: permite impor condições, que todos os grupos devem satisfazer.

Por exemplo, podemos impor uma condição à query que construímos para visualizar o número de clientes por Localidade, por forma a mostrar apenas as Localidades onde a empresa tem mais do que 100 clientes:

```
SELECT Localidade, COUNT(CodTerc)
FROM Clientes
GROUP BY Localidade
HAVING COUNT(CodTerc) > 100
```

Exercícios (bloco 2):

1. Listar clientes apresentando a diferença entre o limite de crédito e o total em débito.
2. Listar artigos indicando a diferença entre o stock máximo e o stock mínimo.
3. Através da tabela de clientes calcule: o total em débito, a média em débito e o valor em débito mais elevado.
4. Liste o número de artigos existentes por família.
5. Liste a média salarial de cada departamento.
6. Apresentar contagem do número de faturas do cliente SOFRIO. Nota: usar apenas tabela *cabecdoc*.
7. Valor de vendas do cliente ALCAD.
8. Apresentar contagem do número de faturas de cada cliente.
9. Liste os departamentos com uma média salarial acima de 750.
10. Liste os clientes com pelo menos 3 faturas, ordenados pelo número de faturas.

Funções Adicionais

Existem algumas funções adicionais que podem ser utilizadas para criar campos calculados em comandos (queries) T-SQL.

Ilustraremos algumas das funções mais significativas disponibilizadas pelo SQL Server, e que podem ser utilizadas em todas as aplicações.

Adicionando Intervalos a datas – DateADD

Exemplo: listar todas as faturas, adicionando trinta dias à sua data.

```
SELECT Num, Data, DATEADD(dd,30,Data) AS DataPag  
FROM Fact
```

Sintaxe: **DATEADD**(Que_Parte, Numero, Data)

Parâmetro	Significado	Valores Admitidos
Que_Parte	Indica a que parte da data a que desejamos adicionar o número especificado.	Ano yy, yyyy Quarto de Ano qq, q Mês mm, m DiaDoAno dy, y Dia dd, d Semana wk, ww Hora hh Minuto mi, n Segundo ss, s Milisegundos ms
Numero	A quantidade a adicionar à parte especificada.	Valor numérico.
Data	Campo do tipo data, ao qual queremos adicionar.	Campo da Base de dados do tipo DateTime ou expressão do mesmo tipo.

A diferença entre duas datas – DateDIFF

Exemplo: Número de dias que ocorreram desde a data de emissão das Faturas.

```
SELECT Num, Data, DATEDIFF(d, Data, Getdate()) AS Num_Dias
FROM Fact
```

Este exemplo usa, adicionalmente, a função **GetDate()**, que retorna a data atual.

Sintaxe: **DATEDIFF**(Que_Parte, DataInicial, DataFinal)

Parâmetro	Significado	Valores Admitidos
Que_Parte	Indica que parte das datas deve ser usada para calcular a diferença.	Os mesmos que em DateADD.
DataInicial DataFinal	As datas que desejamos subtrair.	Dois campos (calculado ou base) ou expressões.

Devolvendo parte de uma data - DatePART

Exemplo: Extrair dia, mês e ano da data de cada fatura, separadamente.

```
SELECT DATEPART(d, Data) AS Dia,
       DATEPART(m, Data) AS Mês,
       DATEPART(yyyy, Data) AS Ano
FROM Fact
```

Sintaxe: **DATEPART** (Que_Parte, Data)

Parâmetro	Significado	Valores Admitidos
Que_Parte	Indica que parte da data se deseja obter	Os mesmos que em DateADD e DateDIFF
Data	A data que desejamos analisar	Campo (calculado ou base) ou expressão

Adicionalmente, foram definidas funções que equivalem a casos concretos de chamadas do DatePart:

DAY(Data)	Equivale a	DatePART(dd, Data)
MONTH(Data)	Equivale a	DatePART(mm, Data)
YEAR(Data)	Equivale a	DatePART(yy, Data)

Arredondamento - ROUND

Arredonda o número/expressão de acordo com a precisão especificada.

Sintaxe: **ROUND** (Num_Expr, Comprimento)

Parâmetros	Significado	Valores Admitidos
Num_Expr	O número ou expressão a arredondar	Campo numérico (base ou calculado) ou expressão
Comprimento	A precisão segundo a qual desejamos arredondar	(ver descrição abaixo)

Valores numéricos possíveis:

- Se for **positivo**, o número/expressão é arredondado para o número de casas decimais especificado por Comprimento (exemplo: 2)
- Se for **negativo**, o número/expressão é arredondado no lado esquerdo do ponto decimal, o número de dígitos especificado por comprimento (exemplo: -1)

Exemplos:

Expressão	Resultado
ROUND(589.8656, 1)	589.9000
ROUND(589.8656, 2)	589.8700
ROUND(589.8656, 3)	589.8660
ROUND(589.8656, -1)	590.0000
ROUND(589.8656, -2)	600.0000
ROUND(589.8656, -3)	1000.0000

Arredondamento por defeito - FLOOR

Retorna o maior valor inteiro menor ou igual que o número/ expressão especificado.

Sintaxe: **FLOOR** (Num_Expr)

Parâmetro	Significado	Valores Admitidos
Num_Expr	O número ou expressão a tratar	Campo numérico (base ou calculado) ou expressão

Exemplos:

Expressão	Resultado
FLOOR(589.8)	589
FLOOR(14.1)	14

Arredondamento por excesso - CEILING

Retorna o menor valor inteiro maior ou igual que o número/ expressão especificado.

Sintaxe: **CEILING** (Num_Expr)

Num_Expr - O número ou expressão a trata. Campo numérico (base ou calculado) ou expressão.

Exemplos:

Expressão	Resultado
CEILING(589.8)	590
CEILING (14.1)	15

Valor Absoluto – ABS

Retorna um número sem sinal, ou seja, converte números negativos para positivos, e mantém os números positivos inalterados.

Sintaxe: **ABS** (Num_Expr)

Num_Expr - O número ou expressão a tratar. Campo numérico (base ou calculado) ou expressão.

Exemplos:

Expressão	Resultado
ABS(-400)	400
ABS (400)	400

Funções Adicionais - Cadeias de Caracteres

O SQL Server disponibiliza algumas funções para o auxiliar na manipulação de cadeias de caracteres (strings). Seguidamente ilustraremos algumas das mais usadas.

Caracteres à esquerda/direita - LEFT, RIGHT

Left - Retorna o número de caracteres especificado do lado esquerdo.

Right - Retorna o número de caracteres especificado do lado direito.

Sintaxe: **LEFT** (Texto, NumCaracteres) ; **RIGHT**(Texto, NumCaracteres)

- **Texto** Cadeia de caracteres a usar. Campo texto ou expressão.
- **NumCaracteres** Número de caracteres a seleccionar. Valor numérico.

Exemplos:

Expressão	Resultado
LEFT('Teste', 2)	'Te'
LEFT('António Manuel',7)	'António'
RIGHT('António Manuel',6)	'Manuel'

Comprimento duma cadeia de caracteres - LEN

Retorna o número de caracteres duma string.

Sintaxe: **LEN**(Texto)

Texto Cadeia de caracteres a usar. Admite um campo do tipo texto ou expressão.

Exemplos:

Expressão	Resultado
LEN('Teste')	5
LEN('António Manuel')	14

Conversão caracteres - LOWER, UPPER

Lower - Retorna texto em minúsculas

Upper - Retorna texto em maiúsculas.

Sintaxe: **LOWER** (Texto) ; **UPPER**(Texto)

Texto Cadeia de caracteres a usar. Campo texto ou expressão.

Exemplos:

Expressão	Resultado
LOWER('Teste')	teste
UPPER('Teste')	TESTE

Retirar espaços - LTRIM, RTRIM

LTRIM - Retira espaços à esquerda

RTRIM - Retira espaços à direita.

Sintaxe: **LTRIM** (Texto) ; **RTRIM**(Texto)

Texto Cadeia de caracteres a usar. Campo texto ou expressão.

Exemplos:

Expressão	Resultado
LTRIM(' 4 espaços à esquerda')	'4 espaços à esquerda'
RTRIM('4 espaços à direita ')	'4 espaços à direita'
LTRIM(RTRIM(' 2 antes e 2 depois '))	'2 antes e 2 depois'

Exemplo prático:

Estas duas funções podem ser usadas para auxiliar a manipulação de campos textuais. Supondo que tem dois campos numa base de dados: Nome e Apelido. Se os campos tiverem sido definidos com uma dimensão de, por exemplo, 30 caracteres, haverá espaços em branco em ambos os campos, para um nome como 'Manuel' e apelido 'Farinhoca'.

É possível usar um campo calculado para construir o nome completo baseado nestes dois campos. Por exemplo:

SELECT Nome + ' ' + Apelido AS NomeCompleto FROM Pessoas

Neste caso, é conveniente que não haja espaços em branco após Nome, de modo a que Nome e Apelido sejam separados apenas por um espaço. Para o efeito basta usar RTRIM(Nome) e LTRIM(Apelido):

```
SELECT RTRIM(Nome) + ' ' + LTRIM(Apelido) AS NomeCompleto
```

Substituir parte de uma string por outra - REPLACE

Sintaxe: REPLACE(Texto, O_Que, Por)

Texto Cadeia de caracteres a usar. Campo texto ou expressão.

O_Que Parte da cadeia de caracteres a substituir. Campo texto ou expressão.

Por Cadeia de caracteres que deve substituir a anterior. Campo texto ou expressão.

Exemplos:

Expressão	Resultado
REPLACE('abcd', 'cd', 'ef')	abef
REPLACE('João Paparoca', 'Paparoca', 'Farinhoca')	João Farinhoca

Funções Adicionais – Agregação

Já lhe falámos anteriormente de algumas funções de agregação. Nomeadamente: AVG, SUM, MAX, MIN e COUNT (Veja “Operadores em campos calculados”).

As funções de agregação fazem um cálculo usando um conjunto de valores, e retornam outro valor. Vejamos seguidamente duas funções de agregação adicionais.

Desvio Padrão - STDEV

Retorna o desvio padrão de todos os valores na expressão indicada.

Sintaxe: STDEV(Campo_ou_Expressao)

Campo_ou_Expressão Campos/valores a utilizar no cálculo Campos/expressões numéricas.

Exemplo:

```
SELECT STDEV(PrecioVenda)
FROM Artigos
```

Variância - VAR

Retorna a variância estatística de todos os valores na expressão indicada.

Sintaxe: **VAR**(Campo_ou_Expressao)

Campo_ou_Expressão Campos/valores a utilizar no cálculo Campos/expressões numéricas.

Exemplo:

```
SELECT VAR(PrecoVenda) FROM Artigos
```

Exercícios (bloco 3):

1. Apresentar lista de faturas da tabela de pendentes cuja data de vencimento já foi ultrapassada e ordenar por data de vencimento.
2. Apresentar lista de faturas da tabela de pendentes cuja data de vencimento já foi ultrapassada em pelo menos 30 dias.
3. Apresentar lista de faturas da tabela de pendentes que vencem nos próximos 60 dias.
4. Apresentar lista de faturas da tabela de pendentes indicando o número de dias de atraso de cada uma delas.
5. Apresentar os anos de serviço dos funcionários.
6. Listar faturas de venda de janeiro de 2015.
7. Contar o número de faturas registadas hoje.
8. Valor de Compras por Fornecedor (resultados sem casas decimais).
9. Lista de artigos com 3 colunas: código de artigo, descrição em letras maiúsculas e o número de caracteres de cada descrição. Ordenar de forma descendente pela 3ª coluna.
10. Listar funcionários cujo nome tem mais de 20 caracteres.

Junção de Tabelas (JOIN)

Em bases de dados relacionais é fundamental poder correlacionar dados armazenados em diferentes tabelas.

Num dos exemplos apresentados anteriormente, temos a informação relativa a clientes numa tabela, e a informação relativa a faturas noutra. Existe, obviamente, uma relação entre Clientes e Faturas. Como se descreveu, o campo CodTerc em Clientes corresponde ao campo CodTerc em Faturas.

Para obtermos a junção dos dados entre as tabelas de Clientes e de Faturas, levamos a cabo a operação de junção, ou JOIN. O objetivo é obter uma tabela final, que inclua os campos de ambas as tabelas (Clientes e Faturas), em que cada linha contém informação dum Cliente e duma Fatura a si associada.

Todas as junções de tabelas são obtidas pelo já descrito comando SELECT.

JOIN Simples - INNER JOIN

A Junção mais simples é obtida através da imposição duma condição ao campo que estabelece a relação entre as duas tabelas. No exemplo que temos vindo a analisar, teremos que impor uma condição de igualdade entre os campos CodTerc da tabela Clientes e da tabela Faturas.

```
SELECT Nome, Num, Data  
  
FROM Clientes, Faturas  
  
WHERE Clientes.CodTerc = Faturas.CodTerc
```

A junção pode englobar mais do que duas tabelas. Supondo que temos uma terceira tabela, com as linhas/detalhes de cada fatura:

Num	CodArtigo	CodLoja	Qtd	...
100	5485	1	5	...
101	4000	1	2	...
102	4205	2	1	...

Fig. 7 Linhas de Fatura (Tabela LFACT)

Supondo também que, além de associar as Faturas aos Clientes, queremos adicionalmente os detalhes de cada Fatura, é fácil constatar que teremos que juntar também as tabelas Faturas e LFact. Estas tabelas estão relacionadas pelo número de fatura (campo Num).



Teremos que indicar o nome da tabela ao incluirmos o campo Num, na lista de campos do SELECT, dado existirem duas tabelas com um campo Num (Fact e LFact). O motor de base de dados necessita de saber de qual das duas tabelas deve extrair o valor para o campo. Se não for especificada qualquer tabela, o servidor retorna uma mensagem do tipo “Ambiguous column name 'Num'.”

Assim, basta incluir também uma condição de igualdade entre estes dois campos nas nossas condições de junção:

```
SELECT Nome, Fact.Num, Data, CodArtigo, Qtd  
FROM Clientes, Fact, LFact  
WHERE Clientes.CodTerc = Faturas.CodTerc  
AND LFact.Num = Faturas.Num
```

Note que o resultado inclui todas as linhas em que os campos de ligação são iguais. Ou seja, um Cliente que não tenha qualquer Fatura associada não vai ser selecionado, dado não haver nenhuma fatura que satisfaça a primeira condição.

Para incluir todos os Clientes, independentemente de terem ou não faturas, usamos o JOIN à esquerda (LEFT JOIN), como veremos na página seguinte.

JOIN Incondicional - CROSS JOIN

O T-SQL disponibiliza uma forma extremamente simples de implementar o produto cartesiano entre duas ou mais tabelas, ou seja, a combinação entre todos os registos duma tabela com os registos doutra(s).

Para mapear todas os Nomes da tabela Clientes com todos os Números da tabela Faturas, basta utilizar o comando seguinte:

```
SELECT Nome, Num  
FROM Clientes, Fact
```

No fundo, equivale a seleccionar os dados de várias tabelas, sem impor qualquer condição de ligação.

JOIN à esquerda - LEFT JOIN

O resultado de um JOIN à Esquerda inclui todas as linhas da tabela à esquerda da cláusula LEFT JOIN, e apenas as linhas correspondentes da tabela à direita. Voltando ao exemplo do JOIN normal que tínhamos definido duas secções atrás, estamos a selecionar Clientes, Faturas e Detalhes de Fatura. Contrariamente ao que acontecia com o JOIN normal (como vimos duas secções atrás, apenas os Clientes com faturas associadas eram selecionados), agora queremos listar TODOS OS Clientes, e os dados das faturas e detalhes a eles respeitantes, se existirem.

A sintaxe é a seguinte:

```
SELECT Campo1 [, Campo2, ...]  
FROM TabelaEsquerda LEFT JOIN TabelaDireita  
ON CampoEsq1 = CampoDir1[, CampoEsq2 = CampoDir2 ...]
```

Retomando o exemplo anterior (Clientes, Faturas e Detalhes das Faturas):

```
SELECT c.Nome, f.Num, f.Data, lf.CodArtigo, lf.Qtd  
FROM Clientes c  
LEFT JOIN Fact f  
ON c.CodTerc = f.CodTerc  
LEFT JOIN LFact lf  
ON f.Num = lf.Num
```

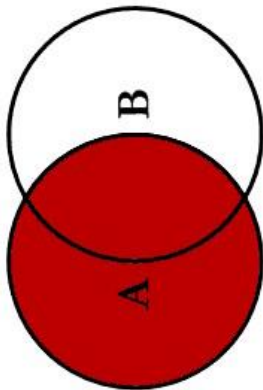
Note que é bastante simples encadear LEFT JOINs. Neste caso, estamos a dizer: "Queremos todos os Clientes, todas as faturas relacionadas com Clientes, e as linhas de fatura relacionados com faturas".

JOIN à Direita - RIGHT JOIN

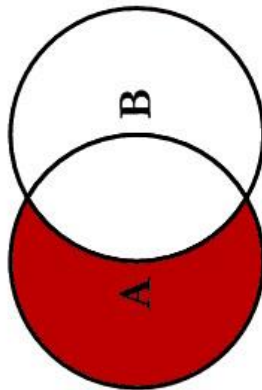
O JOIN à direita é o oposto de um JOIN à esquerda: retorna todas as linhas da tabela à direita da cláusula. Exemplificando com a tabela de países (que contem um código e uma designação por país), e sabendo que a cada Cliente está associado um país, através do campo CodPais, poderíamos selecionar os Clientes e todos os Países, (independentemente de haver Clientes neles ou não), através dum JOIN à direita:

```
SELECT c.Nome, c.CodPais, p.Designacao as Pais  
FROM Clientes c  
RIGHT JOIN Países p  
ON c.CodPais = p.CodPais
```

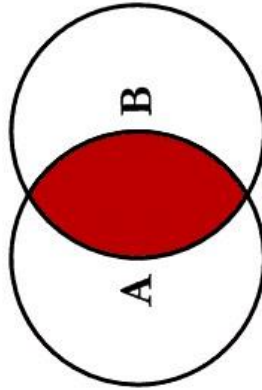
SQL JOINS



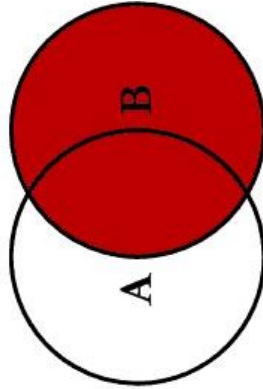
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



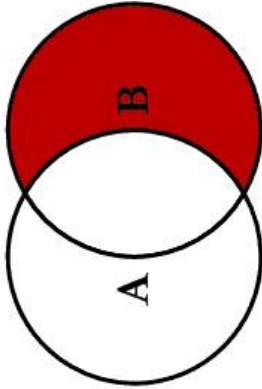
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



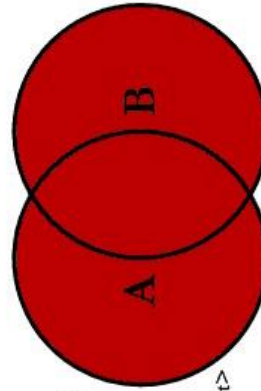
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```


Exercícios (bloco 4):

1. Listar artigos mostrando código e descrição do artigo e também a descrição da família.
2. Lista de todos os clientes apresentando também o nome do vendedor.
3. Lista de clientes com a descrição da condição de pagamento e com a descrição de tipo de terceiro.
4. Validar se existe algum documento de compra que tenha sido criado com condição de pagamento diferente da que consta na ficha do fornecedor.
5. Total de compras por fornecedor, ordenadas de forma decrescente (valor) e superiores a 10.000€. Pretende-se incluir nesta lista o código postal do fornecedor.
6. Total de faturação por localidade de cliente.
7. Listar para cada documento de venda as quantidades totais inseridas do artigo A0001.
8. Somatório do valor de compras por documento de compra do artigo A0001.

União de Tabelas (UNION)

Em variadas situações, desejamos obter uma união entre dois SELECTs distintos. Vejamos um exemplo simples:

Tabela 1		Tabela 2	
Num	Nome	Num	Nome
5	Carlos	6	Alberto
6	Alberto	9	Jerónimo
7	Maria	10	António

Fig. 8 Antes da união

O comando seguinte retorna a união entre as duas tabelas, correspondendo à combinação de resultados de dois comandos T-SQL distintos:

```
SELECT *
FROM Tabela1
UNION
SELECT *
FROM Tabela2
```

O resultado é o seguinte:

Resultado	
Num	Nome
5	Carlos
6	Alberto
7	Maria
9	Jerónimo
10	António

Fig. 9 Resultado da união



Só pode fazer a UNION entre dois SELECTs que retornem o mesmo número de campos, e entre estes deve haver correspondência no tipo de dados. Por exemplo, os comandos seguintes são inválidos.

- O resultado consiste na combinação do conteúdo das duas tabelas.

```
SELECT Nome, Num FROM Tabela1
```

```
UNION
```

```
SELECT Nome FROM Tabela2
```

- Os dois SELECTs têm um número diferente de campos.

```
SELECT Nome FROM Tabela1
```

```
UNION
```

```
SELECT Num FROM Tabela2
```

- O tipo de dados das colunas envolvidas não é compatível (Num é numérico e Nome é textual).

Um exemplo dum comando T-SQL válido é o seguinte, que tem por objetivo retornar o Nome e Código Postal de todos os Clientes de Braga (4700) e Porto (4000):

```
SELECT Nome, CodPostal
```

```
FROM Clientes
```

```
WHERE CodPostal LIKE '4700%'
```

```
UNION
```

```
SELECT Nome, CodPostal
```

```
FROM Clientes
```

```
WHERE CodPostal LIKE '4000%'
```

Estamos neste caso a assumir com o campo CodPostal é textual, e que pode conter "4000 Porto" e "4700 Braga". Assim, justifica-se o uso do operador LIKE.

Intersecção de Tabelas (EXISTS, INTERSECT)

Alguns sistemas de gestão de base de dados disponibilizam a cláusula INTERSECT, para obter a intersecção entre duas tabelas.

Vejamos como podemos obter o Nome de todos os Clientes que tenham faturas associadas, usando a intersecção:

Usando INTERSECT:

```
SELECT Nome
FROM Clientes
INTERSECT
SELECT C.Nome
FROM Clientes C, Faturas F
WHERE C.CodTerc = F.CodTerc
```

Usando EXISTS:

```
SELECT C.Nome
FROM Clientes C
WHERE EXISTS
(SELECT CodTerc
FROM Fact F
WHERE C.CodTerc = F.CodTerc)
```

Diferença entre tabelas (NOT EXISTS)

Usa-se o operador EXISTS negado.

Na secção anterior, usámos o seguinte comando T-SQL para obter todos os Clientes com faturas associadas:

```
SELECT C.Nome
FROM Clientes C
```

WHERE EXISTS

(SELECT CodTerc

FROM Fact F

WHERE C.CodTerc = F.CodTerc)

Podemos usar uma sintaxe semelhante para obter todos os Clientes que não têm faturas associadas:

SELECT C.Nome

FROM Clientes C

WHERE NOT EXISTS

(SELECT CodTerc

FROM Fact F

WHERE C.CodTerc = F.CodTerc)

Conceitos Avançados - Subqueries

Uma Subquery é uma query (comando T-SQL) que retorna um único campo e é incluída dentro de outra query. É possível atribuir à Subquery um nome, que poderá ser usado como qualquer coluna/campo/expressão normal.

Vamos supor que desejávamos saber qual o preço do artigo mais caro em cada fatura.

Podemos usar uma Subquery que nos retorna o preço máximo de entre todos os preços de todos os artigos incluídos em cada fatura, e dar-lhe um nome para que possa ser usada como uma coluna simples.

O resultado será o seguinte:

SELECT f.Num, f.data,

(SELECT MAX(lf.PrecoUni)

FROM LFact lf

WHERE f.Num = lf.Num) AS PrecoUnitarioMax

FROM Fact f

Como podemos observar, a nossa subquery é:

```
SELECT MAX(lf.PrecoUni)  
  
FROM LFact lf  
  
WHERE f.Num = lf.Num
```

Retorna o preço mais elevado encontrado nas linhas de Fatura (LFact) para cada fatura.

Uma SubQuery pode conter dentro de si outras SubQueries, e pode servir de origem de dados para outra query (ver exemplo "Determinar qual a localidade em que a empresa tem mais Clientes" nas próximas seções)

Conceitos Avançados - SubQueries e IN

Como referimos anteriormente, o IN pode ser usado para comparar valores com os resultados duma SubQuery.

Suponha que deseja visualizar todos os clientes que moram na mesma localidade que o Cliente chamado António Fictício. Pode elaborar uma SubQuery que retorne a Localidade onde mora esse cliente, e incluí-la à direita do comando IN:

```
SELECT Nome  
  
FROM Clientes  
  
WHERE Localidade IN  
  
(SELECT Localidade  
  
FROM Clientes  
  
WHERE Nome = 'Antonio Fictício')
```

Outra variante é a utilização da negação em conjunto com o IN (NOT IN). Por exemplo, pode seleccionar todos os Clientes que não moram na mesma localidade que António Fictício.

```
SELECT Nome  
  
FROM Clientes  
  
WHERE Localidade NOT IN  
  
(SELECT Localidade
```

FROM Clientes

WHERE Nome = 'Antonio Fictício')

Conceitos Avançados - Exemplo

Para ilustrar conceitos como SubQueries dentro de outras SubQueries, e o uso de SubQueries como origem de dados de outras, vamos examinar um exemplo concreto:

Determinar: 'qual a localidade em que a empresa tem mais Clientes?'. Como se descreveu anteriormente, a cada Cliente está associada uma Localidade. Como também já vimos antes, é possível determinar facilmente o n.º de Clientes por localidade:

SELECT Localidade, COUNT(CodTerc) AS 'Numero de Cli-entes'

FROM Clientes

GROUP BY Localidade

Neste momento não desejamos uma lista completa de Localidades e Número de Clientes, mas apenas UMA localidade: aquela em que o número de clientes (COUNT(CodTerc)) toma o valor máximo. Ou seja, desejamos selecionar Localidade da tabela de clientes onde COUNT(CodTerc) é máximo.

Qualquer coisa como:

SELECT Localidade

FROM Clientes

GROUP BY Localidade

HAVING COUNT(CodTerc) =

Necessitamos agora de uma Subquery que retorne um único valor, o MAX, de entre todos os que são selecionados pelo nosso comando T-SQL original:

SELECT MAX(NumClientes)

FROM

(SELECT COUNT(CodTerc) AS NumClientes

FROM Clientes

GROUP BY Localidade) AS Q2

Ou seja, estamos a seleccionar o valor máximo de NumClientes da nossa query original (sem o campo Localidade).

Agora, basta-nos juntar os dois comandos:

```
SELECT Localidade
FROM Clientes
GROUP BY Localidade
HAVING COUNT(CodTerc) =
(SELECT MAX(NumClientes)
FROM
(SELECT COUNT(CodTerc) AS NumClientes
FROM Clientes
GROUP BY Localidade) AS Q2)
```

NOTA: Repare que é necessário atribuir um pseudónimo à SubQuery (AS Q2), dado que a estamos a usar como origem de dados na query original (FROM).

Conceitos Avançados – CASE

A expressão CASE permite-nos aplicar a lógica condicional a uma query. Esta lógica condicional fornece uma maneira diferente de adição de blocos de código que podem ser executados, dependendo de uma avaliação booleana da lógica condicional, que resulta em TRUE ou FALSE. Podemos, inclusive, colocar várias expressões condicionais numa única expressão CASE. Para melhor compreendermos a forma como a expressão CASE funciona, vamos mostrar a sintaxe da expressão, e de seguida, mostraremos um exemplo que permite perceber o seu conceito e utilidade no âmbito que se pretende para esta formação de introdução ao T-SQL.

Sintaxe:

```
(CASE WHEN <condição1> THEN <resultado1>
      WHEN <condição2> THEN <resultado2>
      ELSE <resultado contrário> END)
```

Exemplo:


```
SELECT cliente, limitecred, totaldeb,  
(CASE WHEN limitecred = 0 THEN 'limite não definido'  
      WHEN limitecred < totaldeb THEN 'limite excedido'  
      ELSE 'ok' END)  
FROM clientes
```

Esta exemplo devolve uma lista de quatro colunas em que o resultado da quarta coluna é obtido através do CASE. No fundo o que estamos a fazer é a classificar os clientes indicando para cada um se o limite de crédito foi ou não foi excedido. A primeira condição **WHEN limitecred = 0 THEN 'limite não definido'** valida se o limite de crédito foi atribuído ao cliente.

Exercícios (bloco 5):

1. Listar nomes e moradas de funcionários e vendedores.
2. Lista de todos os vendedores que não têm clientes associados.
3. Lista de todos os vendedores com a quantidade de clientes associados, ordenados por essa quantidade e de forma decrescente.
4. Lista com nome de todos os clientes que têm faturas associadas.
5. Lista com nome de vendedores que têm vendas (fatura) associadas.
6. Lista com nome de vendedores que não têm qualquer venda (fatura) associada.
7. Apresentar lista de clientes que não têm qualquer venda (fatura) associada há mais de 9 meses.
8. Apresente para cada vendedor o número de faturas em que ele tem alguma venda, mas considerando apenas aquelas que são de clientes a que ele está associado.
9. Através do CASE classificar homem/mulher da tabela de funcionários.

Manipulação de dados

Como já referimos na secção introdutória, centramos esta formação nos conceitos gerais da linguagem SQL, sua filosofia e aplicabilidade, e no mais universal dos seus comandos, o SELECT, com toda a variedade de conceitos a ele associados.

DML - Data Manipulation Language

Para além do SELECT, existem os seguintes comandos:

- **UPDATE** - Permite atualizar os dados de uma tabela.

Sintaxe:

UPDATE <tabela>

SET <actualização1>, <actualização2>, <actualização3>

WHERE <condições>

Exemplo:

UPDATE Clientes

SET Nome = 'Antonio Manuel Fictício', Morada = 'Alhandra'

WHERE CodTerc = 411

Este exemplo altera para António Manuel Fictício o nome do cliente com código 411, alterando também a sua morada para Alhandra.

- **INSERT** - Permite inserir dados de uma tabela.

Sintaxe:

```
INSERT INTO <tabela>  
(campo1,campo2,campo3,...)  
VALUES  
(valor1,valor2,valor3,...)
```

Exemplo 1:

```
INSERT INTO Clientes  
(CodTerc, Nome, Localidade)  
VALUES  
(9999, 'António Castro', 'Braga')
```

Este exemplo insere na tabela de clientes um novo cliente com código 9999, nome António Castro e localidade Braga.

Exemplo 2:

```
INSERT INTO Clientes  
(Cliente, Nome, Fac_mor)  
SELECT Vendedor, Nome, Morada FROM Vendedores
```

Este exemplo, mais avançado, insere na tabela de clientes todos os vendedores existentes na respetiva tabela. Para o conseguirmos encadeamos o INSERT, onde definimos que campos queremos inserir, com um SELECT que devolve os valores a serem inseridos.

- **DELETE** - Permite apagar registos de uma tabela.

Sintaxe:

DELETE <tabela>

WHERE <condições>

Exemplo:

DELETE Clientes

WHERE CodTerc = 9999

Este exemplo apaga o cliente com código 9999.

Exercícios (bloco 6):

1. Atualizar todos os descontos de cliente de 10% para 16%.
2. Aumentar todos os preços de artigos em 10%.
3. Inserir um novo fornecedor: Código XPTO, Nome XPTO, Morada Alhandra
4. Inserir todos os vendedores na tabela de fornecedores.
5. Apagar o fornecedor XPTO.
6. Apagar todos os idiomas da base de dados DEMO exceto:
 - a. Alemão
 - b. Inglês
 - c. Espanhol
 - d. Italiano
 - e. Português

Anexo – Exercícios

Objetivos

Depois deste capítulo deverá ser capaz de

- Compreender o modelo canónico na prática dominando as principais técnicas de normalização de esquemas relacionais;
- Entender e aplicar as principais instruções de SQL na vertente DML;
- Com base nos exemplos abordados, saber extrapolar para outros casos práticos que possam surgir.

Exercícios de T-SQL

Consultas

Exercício 1

Obtenha uma listagem dos pendentes de Clientes (faturas) ordenada ascendentemente por Entidade e descendentemente por Tipo de Documento e Data de Vencimento. Acrescente uma nova coluna que permita obter o prazo de vencimento, de cada documento, em dias. Altere a listagem por forma a obter o plafond de crédito para cada um desses Clientes.

Exemplo de resultado numa base de dados DEMO:

	entidade	TipoDoc	NumDoc	DataDoc	DataVenc	Prazo	plafond
1	ALCAD	FA	15	2013-01-19 00:00:00.000	2013-02-18 00:00:00.000	30	179681,16
2	ALCAD	FA	14	2012-12-29 00:00:00.000	2012-12-19 00:00:00.000	-10	179681,16
3	ALCAD	FA	5	2012-03-22 00:00:00.000	2012-04-21 00:00:00.000	30	179681,16
4	INFORSHOW	FA	20	2013-02-24 00:00:00.000	2013-03-26 00:00:00.000	30	706,34
5	INFORSHOW	FA	4	2012-03-14 00:00:00.000	2012-04-13 00:00:00.000	30	706,34
6	J.M.F.	FA	17	2013-01-25 00:00:00.000	2013-03-11 00:00:00.000	45	-5864,7
7	J.M.F.	FA	8	2012-06-03 00:00:00.000	2012-07-18 00:00:00.000	45	-5864,7
8	PROPOSTA	FA	10	2012-08-14 00:00:00.000	2012-08-14 00:00:00.000	0	-17924,99
9	SILVA	FA	19	2013-02-10 00:00:00.000	2013-07-10 00:00:00.000	150	130554,54

Exercício 2

Obtenha uma lista de Clientes sem Encomendas. Indique o comando necessário para incluir, nesta lista, os Clientes sem documentos de Venda emitidos. Faça a distinção por meio de uma nova coluna que identifique a situação em cada caso.

Exemplo de resultado numa base de dados DEMO:

Results Messages			
	cliente	nome	situação
7	S.V.M.	Sociedade Vidreira da Marinha, Lda.	Sem Encomendas
8	SOLUCAO-Z	Solução Z-Informática e Serv., Lda	Sem Encomendas
9	SSE	Soluciones de Software de Espanã	Sem Encomendas
10	VD	Cliente Indiferenciado	Sem Encomendas
11	VIDRO-Z	Vidro Z- Vidraria Especializada,Lda	Sem Encomendas
12	LIMA	Empreendimentos do Lima	Sem Doc. Venda
13	NW-CORP.	NW-Electrónica e Sistemas	Sem Doc. Venda
14	S.V.M.	Sociedade Vidreira da Marinha, Lda.	Sem Doc. Venda
15	SSE	Soluciones de Software de Espanã	Sem Doc. Venda

Exercício 3

Em que documentos de Venda se encontra o artigo A0001 e qual a quantidade vendida?

Exemplo de resultado numa base de dados DEMO:

Results Messages		
	tipodoc	Total por tipo Documento
1	AVE	1
2	CBA	1
3	ECL	14
4	FA	24
5	NC	-1
6	ORC	10
7	VD	1

Exercício 4

Elabore uma query que permita saber o valor global esperado nas encomendas em carteira. Classifique o resultado obtido por zonas.

Exemplo de resultado numa base de dados DEMO:

Results Messages		
	descricao	Valor em carteira
1	Comunidade Europeia	1895
2	Grande Lisboa	8554,78
3	Zona Norte	7500

Exercício 5

Visualize o número total de Clientes por Localidade. Aplique uma condição ao agrupamento de forma a visualizar apenas as localidades com 2 ou mais Clientes. Proceda às alterações necessárias para efetuar os mesmos cálculos pelo campo de descrição da Zona.

Exemplo de resultados numa base de dados DEMO:

Results Messages		
	Localidade	Nº clientes
1	Braga	2
2	LISBOA	3
3	PORTO	2

	Zona	Nº clientes
1	Comunidade Europeia	3
2	Grande Lisboa	3
3	Grande Porto	3
4	Zona Norte	2

Exercício 6

Calcule o diferencial entre o limite de crédito e o plafond gasto pelos Clientes. Crie uma nova coluna que permita visualizar 'Boa' ou 'Má' performance consoante a percentagem do plafond gasto seja inferior ou superior a 50%, respetivamente. Efetue as alterações necessárias, de forma a evidenciar as situações em que o cliente já excedeu o Limite de Crédito definido.

Exemplo de resultado numa base de dados DEMO:

Results Messages					
	cliente	nome	limitecred	totaldeb	Performance
1	ALCAD	Soluciones Cad de Madrid, SA	200000	20318,84	Boa
2	INFORSHOW	Inforshow, Informática Comunicação	5000	4293,66	Má
3	J.M.F.	José Maria Fernandes & Filhos, Lda.	0	5864,7	Excedeu
4	LIMA	Empreendimentos do Lima	0	0	N/A
5	MICROAVI	MicroAvi, Inc.	10000	-2363,25	Boa
6	NW-CORP.	NW-Electrónica e Sistemas	300000	0	Boa
7	PROPOSTA	Proposta	0	17924,99	Excedeu
8	S.V.M.	Sociedade Vidreira da Marinha, Lda.	1000	-8586,89	Boa
9	SILVA	Maria José da Silva	250000	119445,46	Boa

Exercício 7

Com base no exercício 4, calcule a percentagem do valor do artigo A0004 em relação ao valor total de cada encomenda.

Exemplo de resultado numa base de dados DEMO:

Results		Messages				
	tipodoc	numdoc	artigo	Total Encomenda	Total do Artigo	% Artigo
1	ECL	3	A0004	1497	1497	100
2	ECL	2	A0004	3573	1497	41,9

Exercício 8

Agrupe as vendas efetuadas por rubrica (ex.: 'REC-VENDAS') e obtenha o valor total obtido. Com base nesta script, obtenha o volume de vendas por zona.

Exclua desta última lista, todas as zonas cujo valor obtido seja inferior a 1000 €.

Exemplo de resultado numa base de dados DEMO:

Results		Messages	
	Rúbrica	Zonas	Total Vendas
1	REC-VENDAS	Angola	3552
2	REC-VENDAS	Comunidade Europeia	18694,9
3	REC-VENDAS	Grande Lisboa	68964,15
4	REC-VENDAS	Grande Porto	4842,36
5	REC-VENDAS	Zona Norte	26506,4

Exercício 9

Elabore uma listagem dos artigos que ainda não foram faturados. Obtenha a descrição ordenada por famílias e apresente o seu stock atual.

Exemplo de resultado numa base de dados DEMO:

Results Messages		
	descricao	stkactual
1	Contabilidade Geral	0
2	Caixote computador base 50*60*25	100
3	Manual do utilizador v.2006	0
4	Prestação de projectos	0
5	Instalação de rede Média	0
6	Transportes de fornecedor	0
7	Montagem Ar Condicionado	0
8	Dissipador de calor série MT	1
9	Processador MT (MT3000)	23

Exercício 10

Obtenha um mapa de resumo de IVA de Fornecedores (compras) com base na tabela de documentos. Para tal, apresente as seguintes colunas: Fornecedor, Tipo Fornecedor, IVA das Mercadorias, IVA de Serviços, (função do fornecedor, tipo de Terceiro e tipo de Artigo).

Exemplo de resultado numa base de dados DEMO:

Results Messages				
	entidade	descricao	Tipo de Terceiro	Valor IVA
1	F0001	Mão de Obra	NULL	782,879975557327
2	F0001	Matéria Prima	NULL	1831,00153801753
3	F0001	Mercadoria	NULL	10739,3741674615
4	F0001	Sub-Produto	NULL	300,000004470348
5	F0002	Matéria Prima	NULL	0
6	F0002	Mercadoria	NULL	3675,28338525221
7	F0003	Matéria Prima	NULL	275,249991461635
8	F0003	Mercadoria	NULL	9119,62771527171
9	F0004	Matéria Prima	NULL	176,903994476795

Exercício 11

Obtenha uma listagem de inventário (valorizada a PCU) onde seja possível obter os seguintes indicadores: stock atual, stock em excesso ou rutura (consoante o caso). Obtenha uma coluna descritiva que deverá apresentar 'excesso' ou 'rutura' em função da situação.

Exemplo de resultado numa base de dados DEMO:

Results		Messages						
	Artigo	Descricao	StkActual	StkMinimo	StkMaximo	Analise	PCUltimo	Valoriza
1	A0001	Pentium D925 Dual Core	20	10	25	Em Conformidade	455	9100
2	A0002	Pentium 4 945+ Dual Core 3.4GHZ	34	20	75	Em Conformidade	926	31484
3	A0003	Processador INTEL CORE 2 DUO E6300 1.86GHZ	49	42	88	Em Conformidade	1198	58702
4	A0004	Processador Intel D326 2.5GHz	26	7	18	Excesso	499	12974
5	A0005	Mesa p/ PC	-81	0	0	Ruptura	62	-5022
6	A0006	Secretária 1,50*0,70*0,70	10	0	0	Excesso	217,99	2179,9
7	A0007	Secretária 1,50*0,70*0,70 c/ Dim	0	0	0	Em Conformidade	0	0
8	A0007.BRANCO	Secretária 1,50*0,70*0,70 c/ Dim(BRANCO)	0	0	0	Em Conformidade	0	0
9	A0007.CASTANHO	Secretária 1,50*0,70*0,70 c/ Dim(CASTANHO)	0	0	0	Em Conformidade	0	0

Inserção, Eliminação e Alteração

Exercício 1

Insira um novo registo de vendedor, sabendo que:

Código: V10

Nome: Vendedor nº 10

Comissão: 3%

Pagamento de Comissões:

20% Liquidação

80% Venda

Chefe dos vendedores:

4 – 3% comissão

5 – 5% comissão

Exercício 2

Introduza para todos os funcionários, a remuneração 'R21' com o valor parametrizado por defeito na ficha de remuneração, para o período de Março de 20XX até Dezembro de 20XX, em que 20XX corresponde ao Ano corrente.

Exercício 3

Altere os dados do cliente XPTO, sabendo que:

Nova morada:

Rua das Fábricas

Novo limite de Crédito: 2500€

Site da empresa: www.xpto.pt

Exercício 4

Atualize a ficha de funcionários para que todos os funcionários referenciem na ficha que recebem Subsídio de Alimentação por dias processados e o valor diário a receber é de 10% sobre o vencimento.