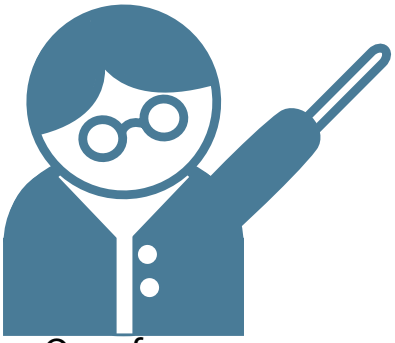
Linguagem SQL

Rodrigo Müller de Carvalho

Graduado e mestrando em Ciência da Computação pelo Instituto de Matemática e Estatística da Universidade de São Paulo. Participação em diversos projetos modelando banco de dados, escrevendo consultas e desenvolvimento de sistemas. Atualmente se especializou em sistemas de Business Intelligence e é desenvolvedor de sistemas analíticos na Universidade de São Paulo e um dos responsáveis pelo desenvolvimento, carga e modelagem do Data Warehouse. Possui experiência como docente de ensino superior na área de tecnologia da informação em disciplinas relacionadas a banco de dados como modelagem de dados, linguagem SQL, sistemas de apoio a decisão e qualidade de dados.



O professor

Capítulo 1
Introdução

1. Introdução

- ✓ Histórico da linguagem SQL;
- ✓ Características da linguagem SQL;
- ✓ Sub linguagens DDL, DML e DQL.

1.1. Histórico da Linguagem SQL

A linguagem SQL foi projetada e implementada no departamento de pesquisa da IBM como a interface para um sistema gerenciador de banco de dados (SGBD) relacional experimental chamado SYSTEM R. Originalmente era chamado de SEQUEL (**S**tructured **E**nglish **Que**ry **L**anguage), um acrônimo para linguagem de consulta em inglês estruturado. Um dos objetivos em seu desenvolvimento era que a linguagem fosse simples de aprender e utilizar. O uso de inglês estruturado na definição da linguagem auxilia a atingir esse objetivo. Por motivos de patentes de nomes não foi possível usar SEQUEL e a linguagem passou a se chamar SQL.

A linguagem SQL se tornou um padrão para banco de dados relacionais, independente de fornecedor, o que facilita a migração de aplicações entre diferentes fornecedores de SGBDs relacionais. A migração é facilitada pois o código SQL escrito seguem os mesmos padrões da linguagem. Além disso podemos escrever programas que acessam mais de um SGBD relacional sem alterar a linguagem de consulta a esses bancos de dados.

A padronização da linguagem SQL é realizado pelos institutos ANSI e ISSO. Os fornecedores de SGBDs relacionais implementam esse padrão e adicionam ao padrão funcionalidades específicas. Ou seja, em um SGBD relacional em particular temos dois conjuntos: as funcionalidades que seguem o padrão estabelecido pelo ANSI/ISO e de funcionalidades particulares desse fornecedor. Quando podemos escolher entre duas funcionalidades em que uma seja padrão e a outra seja particular, uma boa prática é usar a padrão pois facilita uma migração futura e é mais conhecida pelos programadores. O uso de uma particular é feita por motivos de desempenho ou de facilitação de escrita e código, mas deve ser feita de modo consciente.

1.2. Características da Linguagem SQL

A linguagem SQL é uma linguagem declarativa de alto nível. Escrevemos o que queremos de resultado e não como obtemos o resultado. Como exemplo, imagine que você quer que um robô pegue uma garrafa de água gelada para você. Em uma linguagem procedural – aquela que você escreve como obter o resultado, como Python, por exemplo, programaríamos o robô como:

```
1. Vá até o geladeira.
2. Abra a geladeira.
3. Pegue a garrafa com a água.
4. Feche a geladeira.
5. Traga a garrafa de água até mim.
```

Repare que determinamos como o robô executa as ações até a garrafa de água estar em nossas mãos. Em uma linguagem declarativa, determinaríamos para o robô instruções como:

```
1. Traga para mim:
   A garrafa de água
2. Da:
      Geladeira
```

Veja que escrevemos o que queremos e não como o robô executará o procedimento até trazer a água para nós.

Na linguagem SQL escrevemos o que queremos e não como será executada. As otimizações e decisões sobre como executar a consulta são realizadas pelo SGBD.

A base teórica da linguagem SQL são as linguagens formais álgebra relacional e cálculo relacional de tuplas. Essa base teórica é importante pois determina o comportamento e a corretude das instruções da linguagem.

Assim como nomeamos os conceitos de um modelo conceitual, como o modelo entidade relacionamento, e um lógico, como o relacional por exemplo, os conceitos da linguagem SQL também possuem uma nomenclatura. Na linguagem SQL usamos os termos tabela, linha e coluna.

1.3. Sub Linguagens do SQL

A linguagem SQL fornece várias instruções e entre essas instruções temos algumas para definições de dados, consultas e atualizações de dados. As instruções para definições de dados são classificadas como pertencentes a sub linguagem DDL (Data Definition Language – Linguagem de Definição dos Dados). Já as instruções de manipulação de dados são classificadas como pertencentes a sub linguagem DML (Data Manipulation Language – Linguagem de Manipulação de Dados) e por fim classificamos as instruções para consulta de dados na sub linguagem DQL (Data Query Language – Linguagem de Consulta de Dados).

Muitas outras instruções são fornecidas pela linguagem SQL como por exemplo:

- Definição de visões;
- Especificação de segurança e autorização;
- Especificação de controle de transações.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A linguagem SQL é uma linguagem declarativa de alto nível escrita em inglês estruturado que facilita a aprendizagem e uso da linguagem;

- Podemos classificar as instruções da linguagem em sub linguagens, como a linguagem de definição de dados – DDL, a linguagem de manipulação de dados – DML e a linguagem de consulta de dados – DQL;

Capítulo 2
Tipos de Dados

2. Tipos de Dados

- ✓ Armazenamento de dados em SQL;
- ✓ Categorias de tipos de dados;
- ✓ Escolha do tipo de dado.

2.1. Tipos de Dados

Bancos de dados associam tipos de dados a colunas, expressões, variáveis e parâmetros. Os tipos de dados determinam quais os tipos de valores serão permitidos no armazenamento. Todos os dados são armazenados nos bancos de dados em formato de Bytes. Essa é a forma como os computadores trabalham, ou seja, quando irão armazenar a letra A, na realidade, armazenam o código binário “01000001” que a representa. Quando esse dado precisa ser mostrado, o banco de dados traduz o formato binário gravado, na letra A.

A tabela abaixo mostra a tabela ASCII que é uma das tabelas que traduz números binários em caracteres.

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Figura 1 Tabela ASCII - fonte: https://upload.wikimedia.org/wikipedia/commons/d/dd/ASCII-Table.svg

Os tipos de dados podem ser agrupados em categorias de acordo com a tabela abaixo.

Categorias dos tipos de dados
Numéricos exatos
Numéricos aproximados
Data e hora
Cadeias de caracteres (strings)
Cadeias de caracteres Unicode (strings Unicode)
Binários

Outros tipos

A seguir veremos os tipos de dados de acordo com essa classificação.

2.2. Numéricos exatos

A categoria de tipos de dados numéricos exatos armazena números de forma exata, isso é, não fazem a aproximação de um determinado número. Temos três subcategorias de tipos numéricos exatos: inteiro, decimal e monetário.

2.2.1. Inteiro

Nessa subcategoria armazenamos números inteiros como: 1, -50 e 50000. Existem 4 tipos de dados: tinyint, smallint, int e bigint. Cada um deles possui um intervalo de valores que podem ser representados e quanto maior o intervalo, maior o espaço de armazenamento em bytes. A tabela abaixo mostra o intervalo e o espaço de armazenagem em bytes utilizado por cada tipo.

Tipo de dado	Intervalo	Armazenamento (Bytes)
TINYINT	0 a 255	1
SMALLINT	-32.768 a 32.767	2
INT	-2^{31}(-2.147.483.648) a 2^{31} (2.147.483.647)	4
BIGINT	-2^{63} a 2^{63}	8

Ao escolher um tipo de dado inteiro temos que considerar os valores que iremos manipular. Por exemplo, a idade de uma pessoa começa em 0 e o valor máximo é um pouco acima de 100. Então um tinyint é escolhido nesse caso porque todas as idades cabem no intervalo e usamos o menor número de bytes possível. Se escolhermos um int sobraria um intervalo muito grande e estaríamos desperdiçando bytes. A escolha do tamanho do tipo de dado tem impactos sobre o espaço de armazenamento do banco de dados e sobre o desempenho de consultas.

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/int-bigint-smallint-and-tinyint-transact-sql

2.2.2. Decimal

Nessa subcategoria armazenamos números decimais, aqueles que possuem casas decimais, como: 10,56, -789,12346, 5,0. Para representar esses números no SQL precisamos definir dois parâmetros: a precisão do número e a escala. A precisão de um número decimal é o número máximo de dígitos decimais que podem ser armazenados no número, tanto antes ou depois da vírgula. Já a escala é o número máximo de dígitos decimais que poderão ser armazenados à direita da vírgula.

Ao armazenar a nota de um aluno de uma faculdade em que a nota pode variar de 0,00 à 10,00 a precisão necessária é 4 porque o número máximo de dígitos no maior valor (10,00). Já a escala nesse exemplo é 2, pois precisamos armazenar duas casas depois da vírgula.

A tabela abaixo mostra os tipos de dados decimais:

Tipo de dado	Intervalo	Armazenamento (Bytes)
DECIMAL NUMERIC	$-10^{38} + 1$ a $10^{38} - 1$ (quando a precisão máxima é utilizada)	5 à 17, dependendo da precisão

Os tipos de dados decimal e numeric são sinônimos. Para declarar esses tipos de dados podemos fazer de algumas formas:

DECIMAL – Sem especificar precisão e escala são utilizados valores padrões. O padrão é precisão ter o valor 18 e para a escala é 0. Ou seja, é equivalente a DECIMAL (18) ou a DECIMAL (18,0).

DECIMAL (4) – Ao declarar dessa forma, a escala utiliza o padrão 0. É equivalente a DECIMAL (4,0).

DECIMAL (4,2) – Dessa forma temos precisão 4 e escala 2. Informamos tanto a precisão quanto escala. Veja que os números nesse exemplo possuem quatro dígitos no total e duas casas após a vírgula, como em 99,12 e -15,45. Não confundir com 1234,12, que tem 6 dígitos no total e duas casas depois da vírgula. Nesse caso o tipo de dado deve ser pelo menos DECIMAL(6,2).

Atenção: no SQL é utilizado o padrão americano para números, isso é, a separação e milhares é feito por vírgulas e a separação das casas decimais é feita utilizando o ponto. Em português o número 123.456.789,123 é representado como 123,456,789.123. Ou seja, no SQL vamos usar o ponto para indicar as casas decimais, como 123456789.123 ao invés de 123456789,123.

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/decimal-and-numeric-transact-sql

2.2.3. Monetário

Quando necessário armazenar valores monetários temos como opção os tipos money e smallmoney. Como nos inteiros o que varia entre os dois tipos é o intervalo para o valor do número e o espaço de armazenamento necessário. A tabela abaixo mostra os tipos Money e smallmoney detalhando os intervalos e espaço de armazenamento em bytes. Também para esses tipos a escolha de cada um é tomada com base no menor espaço de armazenamento que o número que está trabalhando pode ter. Ou em outras palavras, pensar no valor mínimo e máximo e escolher o tipo de dado em que esses valores se encaixam com menor sobra.

Tipo de dado	Intervalo	Armazenamento (Bytes)
money	-922.337.203.685.477,5808 a 922.337.203.685.477,50807	8
smallmoney	-214.748,3648 a 214.748,3647	4

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/money-and-smallmoney-transact-sql

2.2.4. Cadeias de caracteres

Existem duas subcategorias de tipos de dados categorizados como cadeias de caracteres. Elas são não-Unicode e Unicode. Cadeias de caracteres não Unicode utilizam a codificação de caracteres definida na colação do banco de dados. A codificação basicamente traduz uma sequencia de bytes em um caractere específico, como na tabela ASCII mostrada anteriormente. Codificações comuns são UTF-8 e a ISO-8859-1 (conhecida como Latin-1). Você já deve ter entrado em uma página da Internet e visto caracteres estranhos como ☺ e ☐ no lugar principalmente de caracteres acentuados. Isso provavelmente se deve aos bytes serem armazenados no servidor como ISO-8859-1 e o navegador utilizou outra codificação como UTF-8 para interpretar os bytes. Como as codificações são diferentes um caractere como à foi lido como ☐.

2.2.4.1. Cadeias de caracteres não Unicode

Os tipos de dados nessa subcategoria são char e varchar. Podemos especificar o tamanho de cada tipo deles como char(10) ou varchar(10). O tipo de dado char(n) armazena sempre n caracteres e cada caracteres ocupa 1 byte. Por exemplo, char(10) sempre irá armazenar 10 caracteres, a cadeia 'Ana' é armazenada como ' Ana' e ocupa exatamente 10 bytes, ou seja, é preenchido com brancos no início da cadeia. Já o tipo de dados varchar(n) armazena o número de caracteres que a cadeia possui até o máximo de n. Por exemplo, varchar(10) irá armazenar no máximo 10 caracteres, ou seja, a cadeia 'Ana' será armazenada como 'Ana', sem preencher os 7 espaços até o máximo de 10 caracteres. No varchar, como a cadeia é variável o SQL precisa de 2 bytes extras para marcar o início e o fim da cadeia. A cadeia de caracteres 'Ana' então utiliza mais 2 bytes para essa finalidade e ocupa 5 bytes para armazenamento. A escolha entre os tipos char(n) e varchar(n) é feita com base na cadeia que queremos armazenar, se ela é variável como o nome de uma pessoa, escolhemos o tipo varchar, já que minimizamos o espaço de armazenamento e não temos problemas com os espaços inseridos à esquerda da cadeia, por exemplo, varchar(120). Já se a cadeia sempre tem tamanho fixo, como o CPF de uma pessoa (123.456.789-12) utilizamos o tipo de dado char, no exemplo do CPF utilizaríamos char(14). Note que se no exemplo do CPF se escolhermos varchar(14), para cada valor estaríamos utilizando 16 bytes para cada valor ao invés de 14 bytes usando o char(14).

A tabela abaixo mostra o intervalo de caracteres de cada tipo. Se um varchar for maior que 8000 caracteres, pode-se usar varchar(max) que utiliza até $2^{31} - 1$ caracteres.

Tipo de dado	Intervalo	Armazenamento (Bytes)
char(n)	1 a 8000 caracteres	n bytes sempre
varchar(n)	1 a 8000 caracteres	Número de caracteres da cadeia + 2 bytes (Máximo de n + 2 bytes)
varchar(max)	1 a $2^{31} - 1$ caracteres	Número de caracteres da cadeia + 2 bytes

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/char-and-varchar-transact-sql

2.2.4.2. Cadeias de caracteres Unicode

Ao trabalhar com aplicações que podemos armazenar cadeias de caracteres de várias línguas podemos utilizar os tipos de dados dessa subcategoria. Cadeias de caractere Unicode permitem aos bancos de dados representar e manipular de forma consistente cadeias de caracteres de qualquer sistema de escrita existente (Leia mais sobre Unicode e codificações de caracteres em https://pt.wikipedia.org/wiki/Unicode). Uma coluna definida como Unicode é independente da codificação utilizada na colação do banco de dados, permitindo então utilizar cadeias escritas em qualquer linguagem. A desvantagem desse tipo de dados é que ao invés de utilizar 1 byte por caractere é que utiliza 2 bytes por caractere. Se usarmos um tipo de dado Unicode onde não é necessário, por exemplo, uma cadeia sempre escrita em inglês, estaríamos desperdiçando espaço de armazenamento. Temos os tipos nchar para cadeias de caracteres Unicode de tamanho fixo e nvarchar para cadeias de caracteres Unicode de tamanho variável. O funcionamento desses tipos de dados é o mesmo que os tipos de dados de caracteres não Unicode já discutidos anteriormente. Abaixo a tabela com o intervalo de caracteres de cada tipo. Repare que como são utilizados dois bytes para cada caractere o intervalo é reduzido pela metade.

Tipo de dado	Intervalo	Armazenamento (Bytes)
nchar(n)	1 a 4000 caracteres	n bytes sempre
nvarchar(n)	1 a 4000 caracteres	(Número de caracteres da cadeia * 2) + 2 bytes (Máximo de n + 2 bytes)
nvarchar(max)	1 a $2^{31} - 1$ caracteres	Número de caracteres da cadeia + 2 bytes

Atenção: Ao escrevermos uma cadeia de caractere como 'Ana' o SQL interpreta como não Unicode. Para especificar que a cadeia é Unicode temos que prefixar a cadeia com um N (sempre maiúsculo) como N'Ana' (repare que o N está fora das aspas simples). Sem N no início, o banco irá utilizar a codificação padrão e pode não reconhecer certos caracteres.

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/nchar-and-nvarchar-transact-sql

2.2.5. Data e hora

Para armazenarmos data, hora ou data e hora temos alguns tipos de dados, são eles: date, datetime, datetime2, datetimeoffset, smalldatetime e time. Além do intervalo permitido e do armazenamento em bytes temos também a acurácia de cada tipo. A acurácia é a menor unidade de tempo que um tipo de dados de data e hora pode armazenar. A tabela abaixo contém o tipo de dado, o espaço de armazenamento em bytes, o intervalo de data representado, a acurácia e o formato de entrada recomendado. O formato de entrada recomendado é neutro em relação à linguagem. Isso é, se tivermos o valor '03/02/2018' não sabemos se está representando três de fevereiro de 2018 ou dois de março de 2018. A primeira interpretação é a linguagem português brasileiro e a segunda é verdadeira para a linguagem inglês americano. Então se utilizarmos o valor '20180203', ou seja, formato ano, mês e dia, é neutro em relação à linguagem.

Tipo de dado	Armazenamento (Bytes)	Intervalo de data	Acurária	Formato recomendado
datetime	8	1 de janeiro de 1753 a 31 de dezembro de 9999	3 – 1/3 de milissegundos	'YYMMDD hh:mm:ss:nnn'
smalldatetime	4	1 de janeiro de 1900 a 6 junho de 2079	1 minuto	'YYMMDD hh:mm:ss:nnn'
datetime2	6 a 8	1 de janeiro de 0001 a 31 de dezembro de 9999	100 nano segundos	'YYMMDD hh:mm:ss:nnnnnn'
date	3	1 de janeiro de 0001 a 31 de dezembro de 9999	1 dia	'YY-MM-DD'
time	3 a 5	Não se aplica	100 nano segundos	'hh:mm:ss:nnnnnn'
datetimeoffset	8 a 10	1 de janeiro de 0001 a 31 de desembro de 9999	100 nano segundos	'YY-MM-DD hh:mm:ss:nnnnnnn [+|-] hh:mm'

Tipo de dado	Formatos neutros de linguagem	Exemplos
datetime	'YYYYMMDD hh:mm:ss.nnn' 'YYYY-MM-DDThh:mm:ss.nnn' 'YYYYMMDD'	'20120212 12:30:15.123' '2012-02-12T12:30:15.123' '20120212'
smalldatetime	'YYYYMMDD hh:mm' 'YYYY-MM-DDThh:mm' 'YYYYMMDD'	'20120212 12:30' '2012-02-12T12:30' '20120212'

datetime2	'YYYY-MM-DD' 'YYYYMMDD hh:mm:ss.nnnnnnn' 'YYYY-MM-DD hh:mm:ss.nnnnnnn' 'YYYY-MM-DDThh:mm:ss.nnnnnnn' 'YYYYMMDD' 'YYYY-MM-DD'	'20120212 12:30:15.1234567' '2012-02-12 12:30:15.1234567' '2012-02-12T12:30:15.1234567' '20120212' '2012-02-12'
date	'YYYYMMDD' 'YYYY-MM-DD'	'20120212' '2012-02-12'
time	'hh:mm:ss.nnnnnnn'	'12:30:15.1234567'
datetimeoffset	'YYYYMMDD hh:mm:ss.nnnnnnn [+|-] hh:mm' 'YYYY-MM-DD hh:mm:ss.nnnnnnn [+|-] hh:mm' 'YYYYMMDD' 'YYYY-MM-DD'	'20120212 12:30:15.1234567 +02:00' '2012-02-12 12:30:15.1234567 +02:00' '20120212' '2012-02-12'

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/date-and-time-types

2.2.6. Numérico aproximado

Os tipos de dados numéricos aproximados armazenam valores aproximados de números. Tome cuidado para não utilizar esses tipos de dados quando é necessário exatidão nos números, como em valores monetários, por exemplo. Geralmente esses tipos de dados são utilizados quando são provenientes de medições como peso e distância em que o último número da medida já possui uma aproximação pelo próprio aparelho que realizou a medida. Podemos escolher entre os tipos de dados float e real para números aproximados. A tabela abaixo detalha esses dois tipos.

Tipo de dado	Intervalo	Armazenamento (Bytes)
float(n)	$- 1{,}79 \times 10^{308}$ a $-2{,}23 \times 10^{-308}$, 0 e $2{,}23 \times 10^{-308}$ a $1{,}79 \times 10^{308}$	Depende do valor de n, 4 ou 8 bytes.
real(n)	$- 3{,}40 \times 10^{38}$ a $-1{,}18 \times 10^{-38}$, 0 e $1{,}18 \times 10^{-38}$ a $3{,}40 \times 10^{38}$	4

Consulte! Documentação do SQL Server: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/float-and-real-transact-sql

2.2.7. Binário

Os tipos de dados binários armazenam em cada posição dois valores possíveis: 0 ou 1. Por exemplo 0110011. A tabela a seguir detalha os tipos binary e varbinary.

Tipo de dado	Intervalo	Armazenamento (Bytes)
binary(n)	1 a 8000 bytes	n bytes
varbinary(n)	1 a 8000 bytes	n bytes + 2
varbinary(max)	1 a 2,1 bilhões de bytes (aproximadamente)	tamanho atual + 2

2.2.8. Outros tipos

Existem outros tipos de dados como ilustrado na tabela a seguir. Para conhecer e saber detalhes dos tipos de dados existentes no banco de dados que você utiliza sempre use a documentação do banco de dados, no nosso caso, sempre consulte a documentação do SQL Server, para tipos de dados se encontra no link: https://docs.microsoft.com/pt-br/sql/t-sql/data-types/data-types-transact-sql.

Tipo de dado	Intervalo	Armazenamento (Bytes)	Observações
rowversion	Gerado automaticamente	8	Sucessor do tipo timestamp
uniqueidentifier	Gerado automaticamente	16	Identificador único global (GUID)
xml	0 a 2GB	0 a 2GB	Armazena XML na estrutura hierárquica nativa
cursor	Não aplicável	Não aplicável	Não é um tipo de dado de armazenamento
hierarchyid	Não aplicável	Depende do conteúdo	Representa posição em uma hierarquia
sql_variant	0 a 8000 bytes	Depende do conteúdo	Pode armazenar dados de vários tipos de dados
table	Não aplicável	Não aplicável	Não é um tipo de dado de armazenagem, usado para consulta e operações programáticas

Exercícios

Para cada item abaixo, determine o tipo de dados usando o critério de menor quantidade de bytes possível e justifique sua resposta:

1. Coluna que armazena o número de assentos em um ônibus rodoviário.

2. Coluna que armazena o código de um cliente no Spotify.

3. Coluna que armazena o número de minutos trabalhados por mês de uma pessoa que recebe por hora.

4. Coluna que armazena o nome de uma pessoa.

5. Coluna que armazena a data de nascimento e um cliente.

6. Coluna que armazena o saldo bancário de uma conta.

7. Coluna que armazena o CPF de uma pessoa no formato 123.123.123-12

8. Coluna que armazena a hora de entrada de um funcionário.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para armazenar os dados os bancos de dados utilizam tipos de dados para codificar e decodificar corretamente os bytes armazenados no disco;

- A escolha de um tipo de dado deve ser realizada de modo que o dado seja representável nesse tipo e que diminua a quantidade de bytes necessário para armazenamento e manipulação;

- Um tipo de dado define quais operações podemos realizar nos dados, ao usar um tipo numérico podemos realizar operações como soma e subtração. Já ao escolher um tipo que seja uma cadeia de caracteres podemos realizar operações como concatenação e transformar todos os caracteres em maiúsculo.

Capítulo 3
Criação de Tabelas

3. Criação de Tabelas

- ✓ Comando de criação de tabelas;
- ✓ Chave primária;
- ✓ Chave estrangeira;
- ✓ Unicidade;
- ✓ Determinar valores válidos para colunas;

3.1. Criação de Tabelas

Para persistir os dados em um banco de dados, precisamos armazená-los em objetos do tipo tabela. Comandos de criação, alteração ou eliminação de objetos, fazem parte da categoria de comandos DDL (Data Definition Language – Linguagem de definição de dados). Neste capítulo aprenderemos a sintaxe básica para criação de tabelas e atrelar propriedades nas tabelas, de forma que desempenhem determinadas funcionalidades como verificação de valores, valores padrão e relacionamento entre outras tabelas.

A sintaxe básica para criação de tabelas é mostrada abaixo:

```
USE <database>
GO

CREATE TABLE <nome da tabela>
(
<nome coluna 1> <tipo da coluna> (<tamanho da coluna>) [NOT NULL]
,<nome coluna 2> <tipo da coluna> (<tamanho da coluna>) [NOT NULL]
, ...
);
```

Uma atenção especial deve ser dada em que banco de dados estamos criando a estrutura de dados, portanto usamos o comando USE <database> para posicionarmos na base de dados requerida. O comando anterior é uma simplificação da cláusula CREATE TABLE, sendo que a mesma cláusula aceita inúmeros parâmetros e objetos associados. Como exemplo do comando mostrado, as cláusulas abaixo criam a tabela Aluno na base de dados IMPACTA:

```
USE IMPACTA
GO

CREATE TABLE Aluno
(
  Matricula int
, Nome varchar(20)
, MeioNome varchar(20)
, SobreNome varchar(20)
);
```

O comando anterior posiciona no banco de dados IMPACTA para a criação da tabela Aluno. Essa tabela possui quatro colunas nomeadas Matricula, Nome, MeioNome e Sobrenome. Matrícula armazena números inteiros, Nome, MeioNome e SobreNome armazenam cada um deles uma cadeia de caracteres variável e até 20 caracteres.

3.2. Preenchimento obrigatório de colunas

A cláusula NULL e NOT NULL definem se o preenchimento de determinada coluna é ou não obrigatória. A cláusula NULL é a padrão, ou seja, quando não mencionamos nada como na criação da tabela do exemplo anterior, os campos **permitirão** valores NULL. Se quisermos **obrigatoriamente** que um campo seja preenchido, utilizamos o NOT NULL imediatamente após a definição do tipo de dados da coluna, como no próximo exemplo:

```
CREATE TABLE Veiculo
(
  Placa char(8) NOT NULL
, Marca varchar(20)  NOT NULL
, NomeProprietario varchar(60)
);
```

Esse último exemplo cria uma tabela Veiculo, no banco de dados atualmente em uso, com três colunas de nomes Placa, Marca e proprietário. Repare que indicamos que as colunas Placa e Marca devem possuir sempre um valor. Fizemos isso inserindo NOT NULL após a declaração de tipo de dado de cada coluna.

3.3. Geração automática de valores

Podemos precisar que uma determinada coluna **gere** números automaticamente, como por exemplo, número de matrícula. Os bancos de dados possuem funções específicas para geração de números e podem ser associadas a uma coluna. O SQL Server permite que qualquer tabela tenha um campo com auto numeração, mas apenas uma coluna da tabela pode receber essa funcionalidade. No SQL Server a função se chama IDENTITY e pode receber dois parâmetros: semente e incremento. A semente representará o primeiro número a ser gerado pela série e o incremento significa de quanto em quanto os próximos números serão gerados. Perceba que dependendo do tipo de dados, o IDENTITY não se encaixará. Veja alguns exemplos da função:

```
IDENTITY ou IDENTITY (1, 1) → É o padrão da função (qualquer tipo numérico)

IDENTITY(0, 1) → Qualquer tipo numérico. Inicia no 0 e incrementa de 1 em 1

IDENTITY(-32768, 1) → SMALLINT ou maior. Inicia no -32768, -32767, ..., 0, 1, 2, ...

IDENTITY(255, -1) → TINYINT ou maior. Inicia no 255, 254, 253, ...

IDENTITY(0, 10) → TINYINT ou maior. Inicia no 0, 10, 20, ...
```

Note que por definição, uma coluna auto numerada será automaticamente colocado como NOT NULL, mesmo não escrevendo explicitamente esta cláusula. Os exemplos abaixo mostram a aplicação do IDENTITY:

```
CREATE TABLE Veiculo
(
  idVeiculo INT NOT NULL IDENTITY
, Placa char(8) NOT NULL
, Marca varchar(20)  NOT NULL
);

CREATE TABLE Aluno
(
  Matricula int IDENTITY (500, 1) --Também é NOT NULL!
, Nome varchar(20)
, MeioNome varchar(20)
, SobreNome varchar(20)
);
```

3.4. Chave Primária

Nos exemplos anteriores não definimos nenhuma restrição para não termos linhas duplicadas em uma tabela. Era possível termos duas linhas com informação de um mesmo aluno, por exemplo. Geralmente definimos uma chave primária para nossas tabelas a fim de evitar essa duplicação de linhas. Podemos ter uma e apenas uma chave primária em cada tabela. Por definição a chave primária possui um valor único para todas as linhas de uma tabela. Ou seja, dado um valor correspondente a chave primária de uma tabela, o número máximo de linhas devolvidas é no máximo uma. A chave primária pode ser simples (apenas uma coluna) ou composta (mais de uma coluna).A definição da chave primária (PRIMARY KEY) segue o seguinte modelo:

```
CONSTRAINT <nome da chave primária> PRIMARY KEY (coluna1, coluna2, ...);
```

A tabela aluno pode ser criada com chave primária simples na coluna Matrícula de modo que não existirão duas linhas na tabela com o mesmo valor para Matricula:

```
CREATE TABLE Aluno
(
  Matricula int NOT NULL IDENTITY (500, 1)
, Nome varchar20)
, MeioNome varchar(20)
, SobreNome varchar(20)
, CONSTRAINT pkAluno PRIMARY KEY (Matricula)
);
```

3.5. Chaves estrangeiras

A chave estrangeira faz o relacionamento entre uma ou mais coluna de uma tabela com a chave primária ou única de outra tabela. O formato do comando deve referir as colunas de ambas as tabelas e a tabela onde temos a chave primária.

Uma tabela pode ter várias chaves estrangeiras para outras tabelas, representando os relacionamentos que possui com cada uma das outras tabelas. A sintaxe básica para criação de chave estrangeira é mostrada abaixo:

```
CONSTRAINT <nome da foreign key> FOREIGN KEY (coluna1, coluna2, ...) REFERENCES
<tabela da primary key> (coluna1, coluna2, ...)
```

Um exemplo de chave estrangeira é dado abaixo, onde um aluno realiza provas, e a prova de um aluno contém a matrícula do aluno que a realizou:

```
CREATE TABLE Aluno
(
  Matricula int not null IDENTITY (500, 1)
, Nome varchar(20)
, CONSTRAINT pkAluno PRIMARY KEY (Matricula)
);

CREATE TABLE Prova
(
  idProva int NOT NULL  IDENTITY (1, 1)
, Matricula int NOT NULL
, Nota decimal(4,2) NOT NULL
, CONSTRAINT pkProva PRIMARY KEY (idProva)
, CONSTRAINT fkProva FOREIGN KEY (Matricula)
  REFERENCES Aluno(Matricula)
);
```

3.6. Unicidade

Chave única é semelhante a chave primária, fazendo com que a coluna envolvida seja única na tabela. Podemos ter várias chaves únicas em uma tabela, diferentemente da chave primária, onde só podemos ter uma.

Por exemplo, numa tabela de Cliente, podemos ter um campo que é o número do cliente, CPF e RG. Todos os três campos não permitem repetição na tabela. Podemos eleger qualquer um desses campos como chave primária, por exemplo número do cliente. Neste caso o CPF e o RG poderiam ter chaves únicas, já que a tabela só permite uma chave primária. comando deve referir as colunas de ambas as tabelas e a tabela onde temos a chave primária. A sintaxe básica da restrição de unicidade é mostrada abaixo:

```
CONSTRAINT <nome da unique key> UNIQUE (coluna1, coluna2, ...)
```

Um exemplo da restrição de unicidade é mostrado abaixo:

```
CREATE TABLE Cliente
(
NumCliente int not null IDENTITY (1, 1)
, CPF int NOT NULL
, RG int NOT NULL
, CONSTRAINT pkCliente PRIMARY KEY (NumCliente)
, CONSTRAINT uqClienteCPF UNIQUE (CPF)
, CONSTRAINT uqClienteRG UNIQUE (RG)
);
```

3.7. Valores Padrão

Ao definir uma coluna podemos definir um valor padrão (DEFAULT) que será usado quando não for passado um valor para essa coluna na inserção de dados. Não podem fazer referência a uma outra coluna da tabela, ou a outras tabelas, exibições ou procedimentos armazenados. As definições DEFAULT serão removidas quando a tabela for descartada. A sintaxe básica da restrição é mostrada abaixo:

```
<nome da coluna> <tipo de dados> CONSTRAINT <nome do default>
                          DEFAULT (<valor, texto, data,
                                    função escalar>);
```

Exemplos:

```
MBAExterior VARCHAR(100) CONSTRAINT dfTextoNA DEFAULT 'Não';

Desconto DECIMAL(9, 2) CONSTRAINT dfDesconto DEFAULT 0;

DataVenda DATE NOT NULL CONSTRAINT dfDataVenda DEFAULT(getdate());
```

O exemplo abaixo ilustra a criação de uma tabela Venda usando a restrição de valores padrão:

```
CREATE TABLE Venda
(

DataVenda date not null CONSTRAINT dfDataVenda DEFAULT (getdate())
, Quantidade smallint not null CONSTRAINT dfQtd DEFAULT (1)
, NumeroCliente int not null
, CONSTRAINT pkVenda PRIMARY KEY (DataVenda)
, CONSTRAINT fkVenda FOREIGN KEY (NumeroCliente)
  REFERENCES Cliente(idCliente)
);
```

3.8. Verificação de Valores

Os tipos de dados incluem uma restrição ao preenchimento dos dados em uma coluna, assim, quando definimos uma coluna como TINYINT, sabemos que os valores permitidos irão de 0 a 255.

No mesmo exemplo anterior poderíamos querer que os valores permitidos, além de serem numéricos, pudessem assumir somente os valores de 18 a 90. Esse tipo de restrição pode ser assegurada aplicando regras, que é o significado da cláusula CHECK.

Uma coluna pode ter qualquer número de restrições CHECK e os critérios podem incluir diversas expressões lógicas combinadas com AND e OR. Várias restrições CHECK são validadas na ordem de criação.

A avaliação do critério de pesquisa deve usar uma expressão Booleana (true/false) como base e não pode fazer referência a outra tabela. A restrição CHECK no nível de coluna pode fazer referência somente à coluna restrita. Restrições CHECK oferecem a mesma função de validação dos dados durante instruções INSERT e UPDATE. Se existirem uma ou mais restrições CHECK para uma coluna, todas as restrições serão avaliadas. A forma geral da restrição de verificação é mostrada abaixo:

```
CONSTRAINT <nome da regra> CHECK (<coluna com expressão booleana>)
```

Alguns exemplos são dados abaixo:

```
CONSTRAINT ckIdade CHECK (Idade <= 100)
```

```
CONSTRAINT ckTaxa CHECK (Taxa >= 1 and Taxa <= 5)
```

```
CONSTRAINT CK_emp_id CHECK (emp_id LIKE '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9]
[0-9][FM]' OR emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]')
```

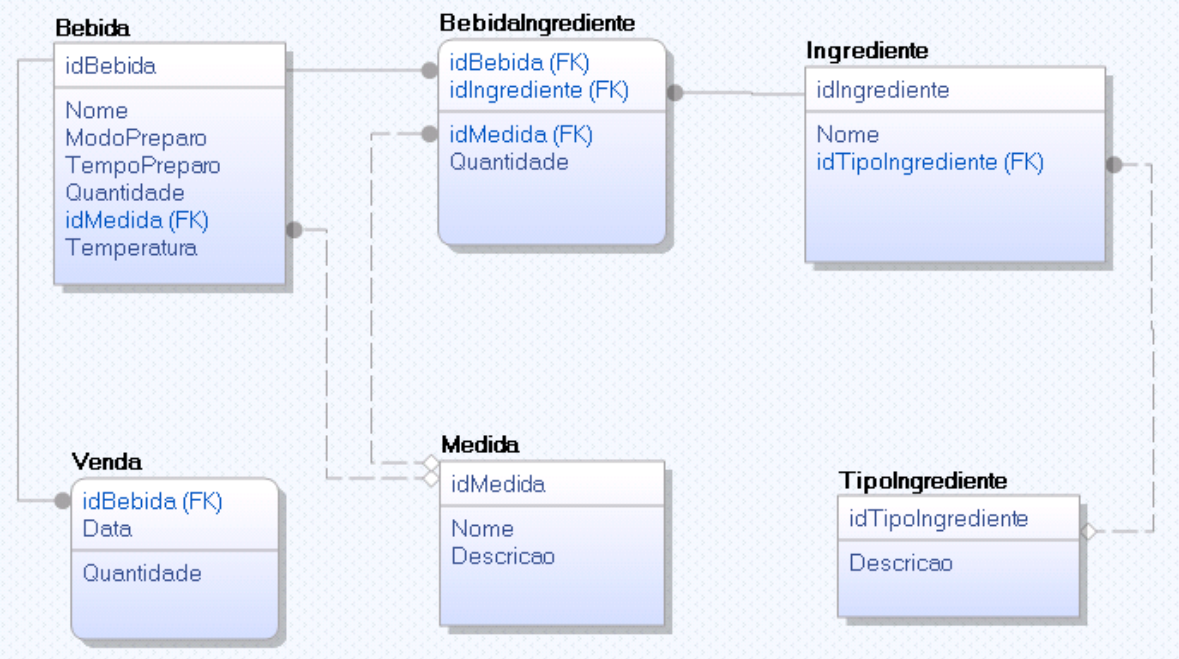
```
CONSTRAINT CK_emp_id CHECK (emp_id IN ('1389', '0736', '0877', '1622', '1756')
OR emp_id LIKE '99[0-9][0-9]')
```

Um exemplo da restrição dentro do comando de criação de tabelas é mostrado a seguir:

```
CREATE TABLE Cliente
(
        idCliente smallint identity(-32767, 1)
      , Telefone VARCHAR(14)
              , DataEntrada datetime
      , Idade tinyint not null,
      , constraint ckIdade CHECK (Idade between 18 and 90)
      , constraint ckTelefone CHECK
            (
           Telefone LIKE '[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]'
           OR
            Telefone LIKE '([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-
9])'
            )
);
```

Exercícios

Crie as tabelas do modelo abaixo escolhendo com cuidado os tipos de dados e respeitando todas as restrições (chave primária, chave estrangeira, unicidade, verificação, default e permissão de valores NULL):



Obs.: Para consulta de seu modelo use SP_HELP.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- CREATE TABLE é o comando que usamos para a criação de tabelas, temos que fornecer um nome para a tabela e nomes de colunas e tipos de dados para o comando executar corretamente;

- Quando precisamos que uma coluna numérica tenha seus valores gerados automaticamente pelo banco de dados usamos IDENTITY. Podemos configurar o número inicial e o incremento de valores.

- A chave primária determina quais colunas determinam que não tenhamos linhas duplicadas em tabelas. Já a restrição de unicidade possui a mesma propriedade que a da chave primária mas adicionamos as outras chaves da tabela. Chaves estrangeiras são determinadas para relacionarmos as colunas de uma tabela com outra.

- Em relação aos valores podemos determinar que uma coluna tenha preenchimento obrigatório usando a cláusula NOT NULL. Para fornecer um valor padrão em caso de ausência na inserção usamos DEFAULT. Para garantir valores válidos como maior que zero usamos CHECK.

Capítulo 4
Modificação e Remoção de Tabelas

4. Modificação e Remoção de Tabelas

- ✓ Modificação de tabelas;
- ✓ Remoção de tabelas;
- ✓ Adição e remoção de restrições.

4.1. Modificação de Tabelas

Podemos alterar tabelas depois de criadas, adicionando novos campos, removendo campos existentes ou alterando tipos de dados e configurações de colunas existentes. O comando básico para isso é o ALTER TABLE. Para adicionar uma coluna, utilizamos a clausula ADD. Para eliminar, DROP COLUMN. Para alterar, ALTER COLUMN. Estes comandos só alteram uma coluna por vez, ou seja, se quisermos adicionar três colunas, teremos que emitir um comando para cada coluna. O comando ALTER TABLE também pertence a sub linguagem DDL.

As sintaxes básicas para alguns comandos de alteração são mostradas abaixo:

```
/* Adiciona uma coluna na tabela */
ALTER TABLE <nome da tabela> ADD <nome da coluna> <tipo de dados>

/* Adiciona uma restrição (chave primária, chave estrangeira, verificação, ...) */
ALTER TABLE <nome da tabela> ADD CONSTRAINT <nome da constraint> ...

/* Altera o tipo de dados de uma coluna da tabela */
ALTER TABLE <nome da tabela> ALTER COLUMN <nome da coluna> <tipo de dados>

/* Remove uma coluna da tabela – remove os dados existentes */
ALTER TABLE <nome da tabela> DROP COLUMN <nome da coluna>

/* Remove uma restrição – veja a importância de nomear as restrições no momento de criá-las */
ALTER TABLE <nome da tabela> DROP CONSTRAINT <nome da constraint>
```

Para alterar a tabela Cliente e adicionar duas novas coluna, a primeira sendo Nome de tipo VARCHAR(30) e a segunda SobreNome de tipo VARCHAR(30) e ter obrigatoriamente valor, usamos os comandos a seguir:

```
ALTER TABLE Cliente ADD Nome varchar(30) NOT NULL;
ALTER TABLE Cliente ADD SobreNome varchar(30) NOT NULL;
```

Se quisermos posteriormente alterar o tipo de dados da coluna Nome da tabela Cliente para uma cadeia de caracteres de tamanho máximo 50, executamos o seguinte comando:

```
ALTER TABLE Cliente ALTER COLUMN Nome varchar(50) NULL;
```

Para remover a coluna SobreNome da tabela cliente, executamos o comando:

```
ALTER TABLE Cliente DROP COLUMN SobreNome;
```

4.2. Remoção de Tabelas

Para eliminar colunas existentes em uma tabela, usamos o comando DROP TABLE.

```
DROP TABLE <nome da tabela1>, <nome da tabela2>, ..., <nome da tabela n>;
```

Note que se uma tabela possui uma chave primária e essa chave participa de relacionamentos com outras tabelas como chave estrangeira, não será permitida a eliminação da tabela que contém a chave primária, ou seja, neste caso, precisamos eliminar as tabelas que mencionam essa chave primária, para depois, conseguirmos eliminar a tabela de chave primária.

Como exemplo, onde temos a tabela Aluno (chave primária), relacionada com a tabela Prova (chave estrangeira), se quisermos eliminar a tabela Aluno, precisamos primeiro, eliminar a tabela Prova, ou retirar as restrições desses objetos, liberando a “amarração” entre eles.

```
DROP TABLE Prova, Aluno;
```

Como exercício, retire as restrições antes de remover as tabelas e as remova na ordem Aluno e depois prova.

4.3. Alteração e Remoção de Restrições

As restrições de chave primária, chave estrangeira, unicidade, valor padrão e verificação podem ser incluídas ou retiradas de uma tabela utilizando os mesmos comandos mencionados anteriormente. Exemplos:

```
ALTER TABLE Cliente Drop Constraint ckIdade;
```

```
ALTER TABLE Cliente Add Constraint ckIdade CHECK (idade between 18 and 90);
```

```
ALTER TABLE Venda Alter Column DataVenda date Constraint dfDtVenda DEFAULT (ge-
tdate());
```

Exercícios

Use o modelo desenvolvido nos exercícios do Capítulo 3 e escreva os seguintes comandos:

1. Retire a chave primária da tabela Venda.

2. Altere o tipo de dado da coluna Descricao da tabela Medida para VARCHAR(100).

3. Insira uma restrição DEFAULT na coluna Descricao na tabela TipoIngrediente com o valor 'Não disponível'.

4. Insira uma restrição CHECK na coluna Quantidade na tabela BebidaIngrediente para a quantidade não ser maior que 100.

5. Insira novamente a restrição de chave primária da tabela Venda com as colunas idBebida e Data.

6. Remova a tabela TipoIngrediente.

7. Insira novamente a tabela TipoIngrediente.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para alterar tabelas usamos ALTER TABLE. Na sequência especificamos o tipo de alteração a ser realizado: adição, remoção e alteração de colunas. Também podemos inserir e remover restrições como chave primária, chave estrangeira, unicidade, verificação, não nulo e valor padrão;

- Para remover uma tabela do banco de dados usamos DROP TABLE. A execução desse comando apaga tanto a tabela quanto os dados que estão na tabela. Ao remover várias tabelas, precisamos nos atentar a ordem da remoção. Primeiro removemos as tabelas que contém as chaves estrangeiras para em seguida a tabela com a chave primária referenciada;

Capítulo 5
Inserção de Dados

5. Inserção de Dados

- ✓ Inserção de uma e várias linhas;
- ✓ Inserção em colunas auto numeradas;
- ✓ Outras formas de inserção.

5.1. Inserção de Dados

Até agora vimos os comandos pertencentes a sub linguagem DDL, isso é, os comandos para definição dos objetos que fazem a persistência dos dados. Com os objetos definidos, precisamos agora conhecer os comandos SQL que manipulem as informações dentro desses objetos. Os comandos que veremos são parte da sub linguagem DML (Data Manipulation Language – Linguagem de Manipulação de Dados). As cláusulas que tratam a inserção, remoção e eliminação de registros dentro de tabelas são INSERT, UPDATE e DELETE, respectivamente. Neste capítulo estudaremos o comando de inserção e nos seguintes a alteração e a remoção.

A declaração INSERT adiciona uma ou mais linhas em uma tabela:

```
INSERT [INTO] tabela_ou_visao [(lista_de_colunas)] valores_de_dados
```

A declaração de INSERT irá inserir um ou mais valores (valores_de_dados) dentro (INTO) da tabela especificada (tabela_ou_visao). A lista_de_colunas é a lista de nome das colunas usadas para especificar as colunas das quais os dados são fornecidos.

5.2. Inserção de uma Linha

Usando a declaração simples de inserção temos os exemplos abaixo:

```
INSERT INTO MinhaTabela (PriKey, Descricao)
VALUES (1, 'TPX450');
```

O comando acima insere uma linha na tabela MinhaTabela com os valores 1 para a coluna PriKey e 'TPX450' para a coluna Descricao.

Outro exemplo usando a declaração simples de inserção é inserir uma nova linha na tabela UnidadeDeMedida com valores 'F2', 'Pés quadrados' e data da inserção – usando a função getdate(). Nesse exemplo não foi fornecido a lista de colunas. Quando essa lista é omitida é utilizada a lista com a ordem de colunas fornecida na criação da tabela:

```
INSERT INTO Producao.UnidadeDeMedida
VALUES ('F2', 'Pés quadrados', GETDATE());
```

5.3. Inserção de Múltiplas Linhas

Podemos inserir várias linhas usando um único comando de inserção, para isso separamos cada linha a ser inserida com uma vírgula, como nos exemplos a seguir:

```
INSERT INTO Producao.UnidadeDeMedida
VALUES ('F2', 'Pés quadrados', GETDATE()),
       ('Y2', 'Jardas quadradas', GETDATE());

INSERT INTO MinhaTabela (PriKey, Descricao)
VALUES (1, 'F200'), (2, 'GTX'), (3, 'CS');
```

5.4. Outras Formas de Inserção

Além das inserções com valores podemos fazer inserção de dados com seleção. Por exemplo, para inserir linhas na tabela MinhaTabela nas colunas PriKey e Descricao usando uma visão de nome MinhaVisao que contém as colunas ChaveEstrangeira e Descricao usamos o seguinte comando:

```
INSERT INTO MinhaTabela (PriKey, Descricao)
      SELECT ChaveEstrangeira, Descricao
      FROM MinhaVisao;
```

Iremos estudar seleção em detalhes em capítulos posteriores bem como visões, após estudar esses capítulos, volte nesse capítulo e aplique o que aprendeu sobre seleção e visões neste capítulo de inserção.

Podemos querer inserir um número determinado de linhas usando uma seleção, por exemplo 5 linhas na TabelaA com linhas da TabelaB, fazemos isso da seguinte maneira:

```
INSERT TOP (1) INTO TabelaA
      SELECT ColunaX, ColunaY
      FROM TabelaB;
```

5.5. Inserção em Tabelas com Colunas Auto Numeradas

Devemos lembrar que colunas com IDENTITY não devem ser mencionadas no INSERT, isso porque os valores dessas colunas são administrados pelo banco de dados e não pelos usuários. Se tentarmos inserir uma linha com um valor para uma coluna auto numerada obtemos um erro, pois esse valor é controlado pelo banco de dados. Considere a tabela criada pelo comando abaixo:

```
CREATE TABLE Veiculo
(
idVeiculo INT IDENTITY(1,1) NOT NULL
, Placa AS char(8) NOT NULL
, Marca AS varchar(20)  NOT NULL
);
```

Comandos de inserção de veículos são ilustrados abaixo, repare que não é passado nenhum valor para a coluna id-Veiculo, que contém a propriedade IDENTITY:

```
INSERT INTO Veiculo (Placa, Marca) VALUES ('XPT-7654', 'Ford');


INSERT INTO Veiculo (Marca, Placa) VALUES ('GM', 'KML-7299');


INSERT INTO Veiculo VALUES ('EXH-2566', 'Fiat');
```

Exercícios

Crie os comandos necessários e na ordem correta para inserir as informações abaixo:

1. 4 Medidas.

2. 6 Ingredientes.

3. 4 Bebidas (cada bebida possuindo no mínimo 2 ingredientes).

4. 3 Tipos de Ingredientes, com 2 ingredientes em cada tipo.

5. Um registro de venda para cada bebida, para cada mês entre fevereiro/2015 e maio/2015.

6.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para inserir uma ou várias linhas em uma tabela usamos INSERT ... VALUES.;
- Não passamos valores para uma coluna auto numerada (com IDENTITY) pois é o banco de dados que administra os valores dessa coluna;
- Durante a inserção de dados podem ocorrer erros quando os valores da inserção violam restrições definidas na criação da tabela, como chave primária que já existe.

Capítulo 6
Remoção de Dados

6. Remoção de Dados

- ✓ Remoção de dados;
- ✓ Selecionar linhas para remoção;
- ✓ Remover as primeiras n linhas;
- ✓ Truncar tabela.

6.1. Remoção de Dados

A remoção de dados é realizada por meio de declaração DELETE. Também faz parte da sub linguagem DML do SQL. A sintaxe para remoção é a seguinte:

```
DELETE tabela_ou_visao
FROM fontes_tabelas
WHERE condicao_de_busca;
```

O comando DELETE irá remover linhas do parâmetro tabela_ou_visao que atender a condição no WHERE (condicao_de_busca). O parâmetro fontes_tabelas pode ser usado para especificar tabelas ou visões adicionais que podem ser usadas na cláusula WHERE. O parâmetro condicao_de_busca é utilizada para selecionar as linhas que serão removidas. Quando condicao_de_busca for avaliada como verdadeira, a linha será removida da tabela_ou_visao.

6.2. Remoção com e sem WHERE

Se não especificarmos uma cláusula WHERE na remoção, todas as linhas da tabela serão removidas. Por exemplo, para remover todas as linhas da tabela Vendedor, usamos o seguinte comando:

```
DELETE FROM Vendedor;
```

Supondo que a tabela Vendedor possui uma coluna de nome registro, se quisermos apagar o vendedor com registro, que é chave primária, de número 10, usamos o seguinte comando:

```
DELETE FROM Vendedor
WHERE registro = 10;
```

Repare que registro sendo chave primária o comando acima remove no máximo 1 linha, se existir linha com registro = 10 ela será removida. Não remove nenhuma outra linha da tabela, pois a condição será falsa para todas as outras linhas.

6.3. Remoção com Sub Consulta

Podemos também realizar remoções de linha utilizando uma sub consulta. Se quisermos remover todas as linhas da tabela histórico de vendas nas quais o salário do vendedor seja maior que 10000,00:

```
DELETE FROM
      HistoricoDeVendas
WHERE
      registroVendedor IN (
            SELECT
                  Registro
            FROM
                  Vendedor
            WHERE
                  Salario > 10000.00
      );
```

6.4. Remoção com TOP

Podemos usar TOP para remover algumas linhas somente da tabela, por exemplo, para remover 2,5 porcento da tabela de estoque, usamos o comando:

```
DELETE TOP (2.5) PERCENT
FROM Estoque;
```

6.5. Truncar tabela

Temos o comando TRUNCATE TABLE para truncar uma tabela, isso é, ao invés de remover todas as linhas de uma tabela usando o comando DELETE sem a cláusula WHERE, podemos usar o comando TRUNCATE TABLE. No entanto, existem diferenças entre os dois comandos:

- Quando executado em uma tabela, reinicia a auto numeração (IDENTITY)
- Não podemos usar TRUNCATE TABLE em tabelas referenciadas pela restrição de chave estrangeira (FOREIGN KEY).
- A declaração TRUNCATE é mais rápida que DELETE.
- Não há como restringir as linhas que serão removidas por meio da cláusula WHERE, diferentemente do comando DELETE.

A sintaxe do comando TRUNCATE TABLE é mostrado a seguir:

```
TRUNCATE TABLE
    [ { database_name.[ schema_name ]. | schema_name . } ]
    table_name;
```

Se quisermos remover todas as linhas da tabela Cliente, que não é referenciada por nenhuma restrição de chave estrangeira, e queremos reiniciar a propriedade IDENTITY de idCliente, podemos usar:

```
TRUNCATE TABLE Cliente;
```

6.6. Boas práticas

Como boas práticas, primeiramente aplicamos o SELECT para verificar se os dados retornados são os que queremos eliminar:

```
SELECT name FROM   Cliente
WHERE  name like 'Marcelo%';
```

Após examinar se a devolução é realmente o que queremos remover, substituímos o SELECT pelo DELETE:

```
DELETE FROM   Cliente

WHERE  name like 'Marcelo%';
```

Exercícios

Use o modelo desenvolvido nos exercícios do Capítulo 3 e populado no Capítulo 05 e escreva os seguintes comandos para a remoção de:

1. Uma medida.

2. Um ingrediente.

3. Uma bebida.

4. Um tipo de ingrediente.

5. Um registro de venda.

Obs.: Após remover, insira novamente os dados removidos no banco de dados.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para remoção de linhas em uma tabela usamos DELETE. Esse comando permite especificar quais linhas serão removidas na cláusula WHERE. Se não especificarmos uma cláusula WHERE todas as linhas da tabela serão removidas;

- A cláusula WHERE recebe uma expressão. Essa expressão deve ser avaliada para verdadeira ou falsa. É aplicada a cada linha da tabela e aquelas que forem avaliadas como verdadeiras serão removidas;

- Usamos TOP(N) para remover as N primeiras linhas da tabela;

- TRUNCATE remove todas as linhas de uma tabela reiniciando propriedades da tabela como IDENTITY. Não é possível especificar cláusula WHERE nesse comando.

Capítulo 7
Atualização de Dados

7. Atualização de Dados

- ✓ Atualização de dados;
- ✓ Selecionar linhas para atualização.

7.1. Sintaxe da Atualização de Dados

A atualização de dados é realizada por meio de declaração UPDATE. Também faz parte da sub linguagem DML do SQL. A sintaxe para atualização é a seguinte:

```
UPDATE tabela_ou_visao
SET nome_da_coluna = expressao
FROM fontes_tabelas
WHERE Condicao_e_busca;
```

A declaração UPDATE altera valores dos dados de uma ou mais linhas de uma tabela. Uma declaração UPDATE referenciando uma tabela_ou_visao pode alterar os dados somente em uma tabela ao mesmo tempo. Possui três cláusulas principais:

- SET – lista de colunas, separados por vírgula, que serão alterados;
- FROM – fornece objetos fonte para a cláusula SET;
- WHERE – Especifica a condição de procura para aplicar as alterações com a cláusula SET.

7.2. Atualização com e sem WHERE

Se não especificarmos uma cláusula WHERE na atualização, todas as linhas da tabela serão atualizadas de acordo com a expressão definida na cláusula SET. Por exemplo, para dar um aumento de 10% (salário reajustado = 1,1 * salário) a todos os vendedores, escrevemos um comando UPDATE da seguinte forma:

```
UPDATE Vendedor
SET Salario = Salario * 1.1;
```

Repare que não especificamos a cláusula WHERE, então todas as linhas são selecionadas para atualização. A cláusula SET especifica que a coluna Salario deve ser atualizada para 1,1 vezes o valor que possui nessa coluna.

Caso o reajuste deva ser dado somente a vendedores que recebem menos de 10000,00 de salário, escrevemos essa condição na cláusula WHERE, para que a atualização seja realizada somente nas linhas que essa condição seja satisfeita:

```
UPDATE Vendedor
SET Salario = Salario * 1.1
WHERE Salario < 10000.00;
```

7.3. Atualização com Sub Consulta

Podemos também realizar remoções de linha utilizando uma sub consulta. Para ilustrar, podemos reescrever a consulta anterior como:

```
UPDATE
      Vendedor
WHERE
      Registro IN (
            SELECT
                  Registro
            FROM
                  Vendedor
            WHERE
                  Salario < 10000.00
      );
```

Exercícios

Use o modelo desenvolvido nos exercícios do Capítulo 3 e populado no Capítulo 05 e escreva os seguintes comandos de atualização:

1. Alterar a(s) venda(s) de fevereiro para 15-janeiro-2015, respectivo as duas últimas bebidas cadastradas.

2. Alterar o nome de um ingrediente.

3. Alterar a quantidade de uma venda.

4. Alterar a descrição de um tipo de ingrediente.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para atualizar dados em uma tabela usamos UPADTE. Esse comando além da cláusula WHERE possui a cláusula SET. Nessa cláusula é que especificamos quais colunas terão seu valor atualizado e qual valor devem ter;

- Se não especificarmos a cláusula WHERE todas as linhas da tabela terão seus valores atualizados de acordo com a especificação no SET;

- Podemos utilizar uma sub consulta para determinar quais linhas serão atualizados na cláusula WHERE;

- Podemos usar diversas funções do SQL Server na atualização e na seleção de linhas, como MAX, MIN, AVG, UPPER, etc.

Capítulo 8
Seleção de Dados

8. Seleção de Dados

- ✓ Seleção de dados;
- ✓ Ordem de execução da seleção;
- ✓ Apelidos em colunas e tabelas;
- ✓ Linhas distintas no resultado;
- ✓ Tratamento de nulos;
- ✓ Ordenação do resultado.

8.1. Seleção de Dados

Já aprendemos os comandos principais das sub linguagens DDL e DML da linguagem SQL. Ou seja, somos capazes de definir as tabelas necessárias para uma aplicação, escolher corretamente os tipos de dados de cada coluna, garantir que nosso modelo siga o modelo relacional por meio das restrições – chave primária, chave estrangeira, obrigatoriedade de preenchimento, unicidade, verificação de valores. Alterar esse esquema de acordo com a evolução da aplicação usando comandos de alteração de esquema e também manipular os dados usando comandos de inserção, remoção e alteração de dados.

Agora iniciamos o estudo da sub linguagem de consulta de dados – DQL (Data Query Language) em que o comando principal é o SELECT. O SELECT é uma declaração SQL que retorna um conjunto de resultados de linhas de uma ou mais tabelas. Ele recupera zero ou mais linhas de uma ou mais tabelas-base, tabelas temporárias, funções ou visões em um banco de dados. Também retorna valores únicos de configurações do sistema de banco de dados ou de variáveis de usuários ou do sistema.

Na maioria das aplicações, SELECT é o comando mais utilizado. Como SQL é uma linguagem não procedural, consultas SELECT especificam um conjunto de resultados, mas não especificam como calculá-los, ou seja, a consulta em um plano de consulta é deixada para o sistema de banco de dados, mais especificamente para o otimizador de consulta, escolher a melhor maneira de retorno das informações que foram solicitadas. Ou seja, escrevemos o que queremos que seja devolvido.

8.2. Cláusulas do Comando SELECT e Ordem de Execução

As cláusulas do comando SELECT são as seguintes:

- SELECT: Define quais as colunas que serão retornadas;
- FROM: Define a(s) tabela(s) envolvidas na consulta;
- WHERE: Filtra as linhas requeridas;
- GROUP BY: Agrupa a lista requerida (utiliza colunas);
- HAVING: Filtra as linhas requeridas, pelo agrupamento;
- ORDER BY: Ordena o retorno da lista.

De acordo com a consulta que queremos realizar no banco usamos as cláusulas necessárias, ou seja, não usamos todas em todas as consultas. No entanto a ordem de escrita das cláusulas segue a ordem especificada acima. A ordem como a consulta (query) é escrita, não significa que será a mesma ordem que o banco de dados utilizará para executar o processamento que é a seguinte:

- FROM: Primeiro as tabelas necessárias;
- WHERE: Depois é realizada a filtragem dessas linhas e condições de junções;

- GROUP BY: Após termos as linhas de interesse, podemos agrupá-las;
- HAVING: Com os grupos formados, pode-se filtrar por grupos;
- SELECT: Após filtrar os grupos, pode-se selecionar as colunas desejadas na devolução da consulta;
- ORDER BY: Por fim, podemos ordenar os valores das colunas que serão devolvidas.

8.3. Exemplos de Seleção Simples

A forma mais simples da declaração SELECT é a utilização junto ao elemento FROM, conforme mostrado abaixo. Note que na <lista de seleção> é realizada uma filtragem vertical, ou seja, devolve uma ou mais colunas de tabelas, mencionadas pela cláusula FROM.

- SELECT: <lista de seleção>
- FROM: <tabela fonte>

Exemplo:

```
SELECT Nome, Sobrenome
FROM   Cliente;
```

O exemplo anterior devolve todas as linhas da tabela Cliente (pois não foi realizada nenhuma filtragem de linhas utilizando a cláusula WHERE) e as colunas Nome e Sobrenome (pois foram especificadas na cláusula SELECT).

Outros exemplos:

```
--( * ) - Retorna todas as colunas da tabela exemploSQL
SELECT  * FROM exemploSQL


/*( coluna ) - Retorna a coluna texto_curto_naonulo da tabela exemploSQL */
SELECT  texto_curto_naonulo FROM exemploSQL


/*
(coluna 1, coluna 2, ...) - Retorna as colunas texto_curto_naonulo e numero_check
da tabela exemploSQL
*/
SELECT  texto_curto_naonulo, numero_check FROM exemploSQL
```

8.4. Utilizando Operadores Matemáticos na Seleção

Podemos fazer utilização de diversos operadores matemáticos para cálculo de valores, abaixo mostramos os principais operadores.

Operador	Descrição
+	Adição ou concatenação
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo

O exemplo abaixo devolve as colunas preco e qtd da tabela DetalhesDoPedido e o valor do pedido, que é uma multiplicação do preço e da quantidade:

```
SELECT preco, qtd, (preco * qtd)
 FROM   DetalhesDoPedido;
```

Outros exemplos:

```
SELECT  20 + 20 / 5 FROM exemploSQL
SELECT  (20 + 20) / 5 FROM exemploSQL
SELECT  20 + (20 / 5) FROM exemploSQL
SELECT  ( (10+2) / 2 ) * 0.3 ) % 2
SELECT  Nome, Salario * 1.07 FROM Funcionario


/* Nota: O operador + se transforma em concatenador quando lidamos com string */
SELECT  'Hoje' + ' ' + 'é' + ' terça-feira ' + 'ou' + ' quinta-feira '
```

8.5. Apelidos em Colunas e Tabelas

Pode ser necessário darmos apelidos (Aliases) a colunas para facilitar o entendimento no retorno dos dados, no exemplo de detalhes de pedidos, ao realizar a conta preço vezes quantidade total não especificamos um apelido para a coluna, ao executar no SQL Server, a coluna será devolvida como sem nome. Para dar um apelido a coluna fazemos:

```
SELECT idPedido, preco, qtd AS Quantidade
 FROM   DetalhesDoPedido;
```

Também podemos realizar a mesma operação com =:

```
SELECT idPedido, preco, Quantidade = qtd
 FROM   DetalhesDoPedido;
```

Ou mesmo sem a necessidade do AS:

```
SELECT idPedido, preco ValorProduto
 FROM   DetalhesDoPedido;
```

Também pode ser necessário darmos apelidos em tabelas, principalmente quando formos realizar junções (joins):

```
/* Apelidos em tabelas com a cláusula AS: */
SELECT  idPedido, dataPedido
 FROM      Pedido AS SO;

/* Apelidos em tabelas sem a cláusula AS: */
SELECT  idPedido, dataPedido
 FROM      Pedido  SO;

/* Usando os apelidos no SELECT */
SELECT  SO.idPedido, SO.dataPedido
 FROM      Pedido AS SO;
```

8.6. Linhas Repetidas

Ao executar uma consulta como:

```
SELECT pais
FROM   Cliente;
```

Um resultado possível de ser obtido é o seguinte:

```
pais
--------------
Argentina
Argentina
Áustria
Áustria
Bélgica
Bélgica
```

Como país é uma coluna que não é chave primária e nem única, várias linhas podem ter o mesmo valor para essa coluna. Então ao realizar a consulta acima é esperado que tenhamos linhas duplicadas. Se for necessário eliminar as linhas repetidas podemos aplicar a cláusula DISTINCT, que retira repetições de linhas para todas as colunas descritas na declaração SELECT:

```
SELECT DISTINCT pais
 FROM   Cliente;
```

E a saída será:

```
pais
--------------
Argentina
Áustria
Bélgica
```

8.7. Devolução de Somente Algumas Linhas

Muitas vezes queremos visualizar apenas o retorno de algumas linhas e não necessariamente todos os registros de uma tabela. Podemos utilizar a cláusula TOP para isso:

- TOP(N): Retorna uma certa quantidade de linhas definido;

- TOP(N) PERCENT: Retorna um certo percentual de linhas definido.

Exemplos:

```
/* Devolve 10 linhas da tabela exemploSQL */
SELECT top 10 * FROM exemploSQL;

/* Devolve 10% das linhas da tabela exemploSQL */
SELECT top 10 percent * FROM exemploSQL;
```

8.8. Selecionando as Linhas a Serem Devolvidas

A cláusula WHERE faz o filtro horizontal em uma consulta, ou seja, permite uma redução do número de linhas que retornarão na consulta. Operadores são utilizados para avaliar uma ou mais expressões que retornam os valores possíveis: TRUE, FALSE ou UNKNOWN. A devolução de dados se dará em todas as linhas onde a combinação das expressões retornarem TRUE.

Operadores de comparação escalar: =, <>, >, >=, <, <=, !=.

Exemplo:

```
SELECT PrimeiroNome, NomeMeio, UltimoNome
FROM Pessoa
WHERE DataNascimento >= '20040101'
```

O exemplo acima devolve somente as linhas em que o valor para DataNascimento seja maior que primeiro de janeiro de 2004, e não todas as linhas da tabela pessoa. Após isso é feita a seleção das colunas representando o primeiro nome, nome do meio e último nome.

Outros exemplos usando cláusula WHERE simples são dados abaixo:

```
SELECT IdEntidadeNegocio AS 'Número Identificação Empegado',
       DataContratacao,
       HorasDeFerias,
       HorasDoente
FROM    RecursosHumanos.Empregado
WHERE IdEntidadeNegocio <= 1000

SELECT
      PrimeiroNome,
      SobreNome,
      Telefone
FROM
      Pessoa.Pessoa
WHERE
      PrimeiroNome = 'Jhon'
```

Podemos usar operadores lógicos para combinar condições na declaração:

```
/* Retorna somente registros onde o primeiro nome for 'John' E o sobrenome for
'Smith' */
WHERE PrimeiroNome = 'John' AND UltimoNome = 'Smith'

/* Retorna todos as linhas onde o primeiro nome for 'John' OU todos onde o so-
brenome for 'Smith' */
WHERE PrimeiroNome = 'John' OR UltimoNome = 'Smith'
```

```
/* Retorna todos as tuplas onde o primeiro nome for 'John' e o sobrenome NÃO for
'Smith' */

WHERE PrimeiroNome = 'John' AND NOT UltimoNome = 'Smith'
```

Nem sempre usamos operadores de comparação. Em algumas situações podemos usar outros operadores que são chamados de predicados, simplificando a escrita do código. Alguns exemplos de predicados em SQL são: IN, BETWEEN, ANY, SOME, IS, ALL, OR, AND, NOT e EXISTS.

Por exemplo, se quisermos devolver todas as linhas da tabela Pessoa onde endereço de email não seja nulo, utilizamos o predicado IS NOT NULL (é não nulo):

```
SELECT
      PrimeiroNome,
      SobreNome,
      Telefone
FROM
      Pessoa.Pessoa
WHERE
      EnderecoEmail IS NOT NULL
```

O predicado BETWEEN restringe os dados por meio de uma faixa de valores possíveis especificada pelo valor inicial e o valor final. Para devolver todos os pedidos entre as datas de 01 de janeiro de 2011 e 31 de agosto de 2011 podemos escrever:

```
SELECT
      DataPedido,
      NumeroConta,
      SubTotal,
      Impostos
FROM
      Pedidos
WHERE
      DataPedido BETWEEN '20110801' AND '20110831'

/*
É equivalente a substituir o BETWEEN pela expressão:
DataPedido >= '20110801' AND DataPedido <= '20110831'
*/
```

O predicado IN usa uma lista de possibilidades de valores que podem atender a consulta. Se quisermos devolver os pedidos que tenham o valor de IdProduto igual a 750, 753, 765 ou 770 podemos escrever a consulta como:

```
SELECT
      DataPedido,
      NumeroConta,
      SubTotal,
      Impostos
FROM
      Pedidos
WHERE
      IdProduto IN (750, 753, 765, 770)

/*
É equivalente a substituir a linha do WHERE usando IN por:
IdProduto = 750 OR IdProduto = 753 OR IdProduto = 765 OR IdProduto = 770
*/
```

8.9. Usando LIKE para colunas de cadeias de caracteres

O predicado LIKE permite realizar consultas mais refinadas em colunas do tipo cadeia de caracteres (char, varchar, ...). Usamos para verificar padrões dentro de campos cadeia de caracteres e utiliza símbolos, chamados de coringas, para permitir a busca desses padrões. Os principais tipos coringa são:

- % (Porcentagem) representa qualquer cadeia de caracteres e qualquer quantidade de caracteres. Exemplo: LIKE 'Carol%' é verdadeira para cadeias como Carolina, Caroline e Carola;

- _ (Underscore) representa qualquer caractere, mas apenas um caractere. Exemplo: LIKE 'Carol_' é verdadeira para cadeias como Carola, mas não Carolina nem Caroline;

- [<List of characters>] representa possíveis caracteres que atendam a cadeia procurada. Exemplo: LIKE 'Carol[ao]' é verdadeira para as cadeias Carola e Carolo, somente.

- [<Character> - <character>] representa a faixa de caracteres, em ordem alfabética, para a string procurada. Exemplo: LIKE 'Carol[a-e]' é verdadeira para as cadeias Carola, Carolb, Carolc, Carold e Carole somente.

- [^<Character list or range>] representa o caractere que não queremos na pesquisa. Exemplo: LIKE 'Carol[^o]' é verdadeira para todas as cadeias que tenham Carol no início e mais um caractere exceto 'o', ou seja, é falso para Carolo e verdadeiro para Carola, Carolb, etc.

8.10. Utilização do NULL

O NULL representa ausência de valor ou valor desconhecido. Nenhuma das sentenças abaixo é verdadeira porque o banco de dados não pode comparar um valor desconhecido com outro valor que ele também não conhece:

```
NULL = 0 -- Resultado é desconhecido (Não é verdadeiro!)

NULL = '' (branco ou vazio) -- Resultado é desconhecido (Não é verdadeiro!)

NULL = 'NULL' (cadeia NULL) -- Resultado é desconhecido (Não é verdadeiro!)

NULL = NULL -- Resultado é desconhecido (Não é verdadeiro!)
```

Para trabalhar com valores NULL, temos que utilizar os predicados IS NULL e IS NOT NULL para a lógica da consulta estar correta. Predicados retornam o valor desconhecido quando comparados com valores desconhecidos (valores faltando), ou seja, não são retornados na consulta. Por exemplo:

```
SELECT IdConsumidor, Cidade, Estado, Pais
FROM Vendas.Consumidor
WHERE Estado IS NOT NULL;
```

8.11. Ordenação dos Resultados da Consulta

Por padrão ao realizarmos uma consulta não existe garantia da ordem de devolução. Por mais que possa parecer que ao executarmos a consulta repetida vezes o resultado seja ordenado não há essa garantia. Ao adicionarmos um novo índice a uma tabela, por exemplo, a ordem de devolução da consulta pode se alterar. Se precisamos garantir uma ordem específica na devolução de uma consulta, por exemplo, listar os consumidores por estado em ordem alfabética crescente, temos que especificar a cláusula ORDER BY:

```
SELECT IdConsumidor, Cidade, Estado, Pais
FROM Vendas.Consumidor
WHERE Estado IS NOT NULL
```

```
ORDER BY Estado; -- Poderíamos adicionar ASC ao final
```

Usamos as cláusula ASC e DESC após cada campo do comando ORDER BY. A ordenação ASCendente é a padrão quando não mencionamos explicitamente. Quando é necessário a ordenação DESCendente, usamos a cláusula DESC:

```
SELECT IdConsumidor, Cidade, Estado, Pais
FROM Vendas.Consumidor
WHERE Estado IS NOT NULL
ORDER BY Estado DESC;
```

Exercícios

Use o modelo desenvolvido nos exercícios do Capítulo 3 e populado no Capítulo 05 e escreva os seguintes comandos:

1. Selecionar a quantidade de bebidas com tempo de preparo entre 5 e 10 minutos.

2. Qual a soma de vendas para 3a. bebida cadastrada, entre fevereiro e março.

3. Liste todos os nomes de bebidas, nomes dos ingredientes, nomes das medidas e quantidades, ordenadas na respectiva sequência das colunas solicitadas.