

ETEC-SP – Escola Técnica Estadual Parque da Juventude

APOSTILA DE

Tecnologia Linguagem Banco de
Dados II

TLBD II

Professor: Osvaldo Rodrigues Sérgio

SUMÁRIO

Índice	Página
Alias	36
Cláusula CROSS JOIN	97
Cláusula GROUP BY	113
Cláusula GROUP BY / HAVING	118
Cláusula GROUP BY / WITH ROLLUP	119
Cláusula GROUP BY com JOIN	120
Cláusula INNER JOIN	91
Cláusula JOIN	91
Cláusula LEFT JOIN	93
Cláusula ORDER BY ASC / DESC	60
Cláusula RIGHT JOIN	93
Cláusula TOP	63
Cláusula UNION ALL	100
Cláusula WHERE	46
Cláusula WHERE com o operador BETWEEN / NOT BETWEEN	52
Cláusula WHERE com o operador IN / NOT IN	50
Cláusula WHERE com o operador LIKE / NOT LIKE	53
Cláusula WHERE com os operadores AND e OR	47
Coluna Virtual	37
COMMIT	128
Conexão banco de dados	04
Constraints	72
Constraints – Regras	75
CREATE DATABASE	30
CREATE TABLE	30
DataTypes	34
DELETE	46
Distinct	37
DROP TABLE	31
Formato de inserção de data	34
INSERT	31
INSERT com SELECT	37
INSERT declarativo	39
INSERT posicional	39

Instalar Microsoft SQL Server	04
Integridade e consistência dos dados	64
Linguagem SQL	123
Linguagem SQL – Comando DQL	123
Linguagem SQL – Comandos DCL	123
Linguagem SQL – Comandos DDL	123
Linguagem SQL – Comandos DML	123
PROCEDURE	131
ROLLBACK	128
SELECT	31
SubQueries com igualdade (=)	108
SubQueries com IN / NOT IN	106
SubQuery	104
Tabela DELETED	130
Tabela INSERTED	130
Tipos de relacionamentos (cardinalidade)	76
Trabalhando com o MS SQL Server	29
Transações	126
TRIGGERS	129
UPDATE	45
USE	30
Bibliografia	132

Conexão ao banco de dados:

Neste curso, utilizaremos como banco de dados de apoio o MS SQL SERVER 2005.

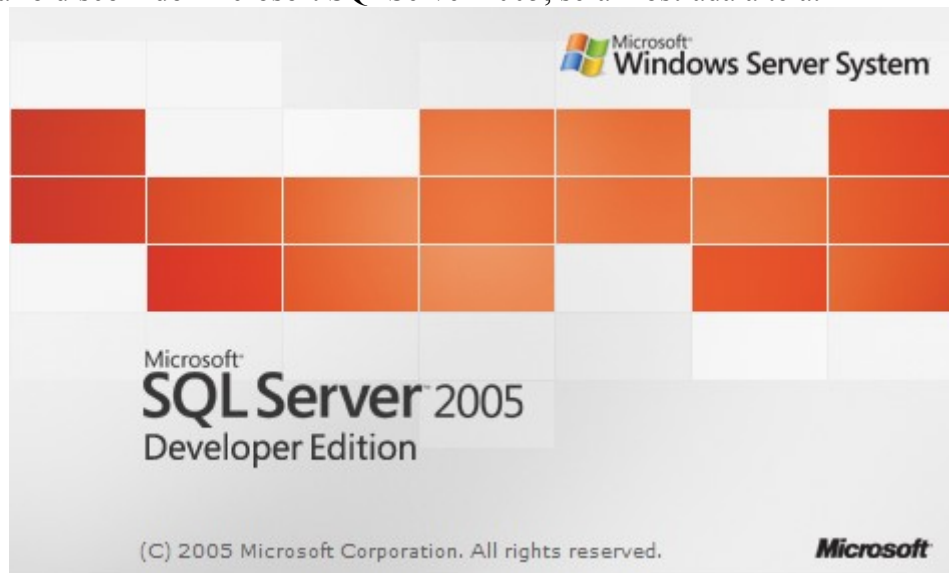
Para instalar o MS SQL Server 2005:

Para a instalação do Microsoft SQLServer 2005, deverá atender aos seguintes requisitos:

- Windows 2000/XP/2003/Vista (após a instalação precisa do service pack 1)
- Espaço em disco de 4Gb
- Discos de instalação do Microsoft SQLServer 2005 (disco 1 e disco 2)
- Paciência

A seguir, serão mostradas as telas quando da instalação.

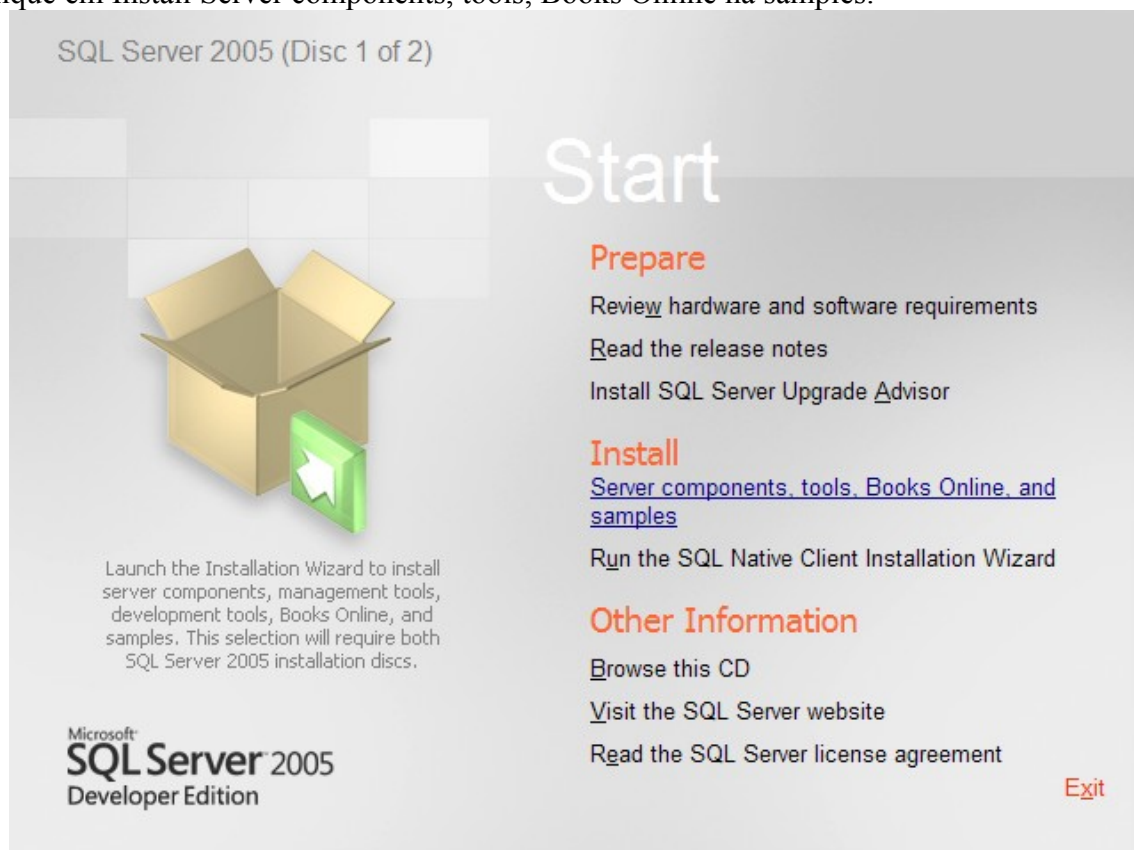
Ao colocar o disco 1 do Microsoft SQLServer 2005, será mostrada a tela:



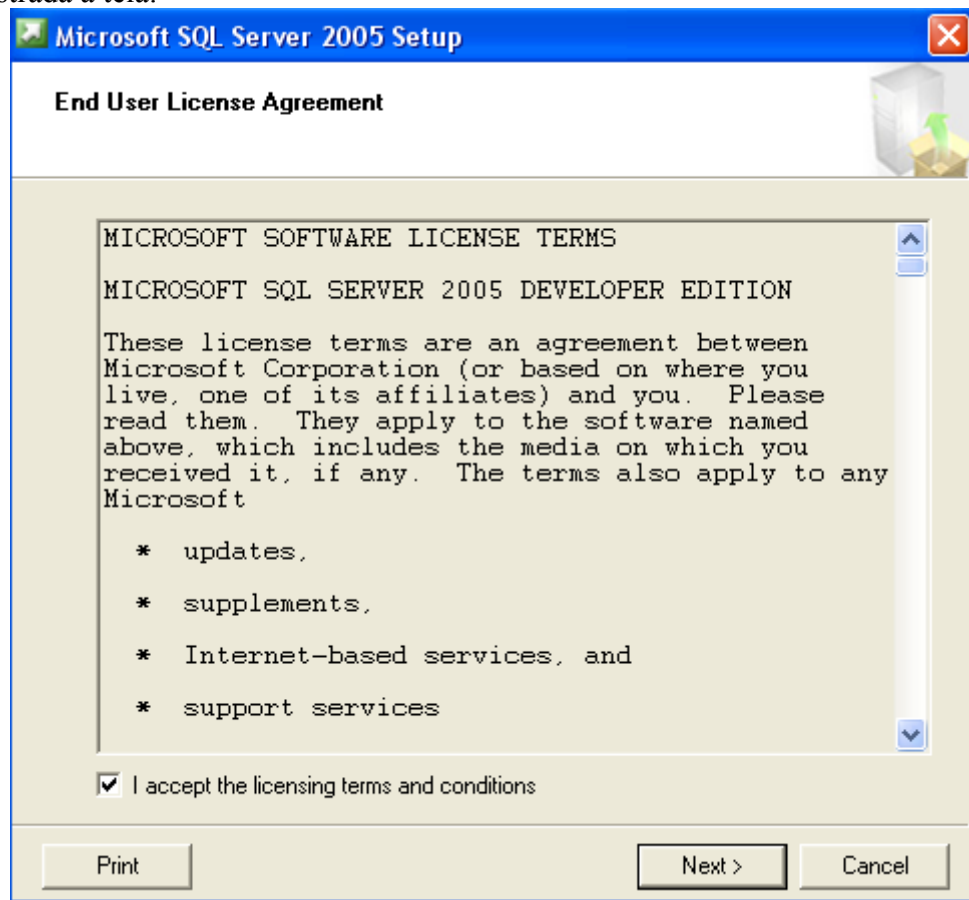
E logo após, a tela:



Clique em Install Server components, tools, Books Online na samples.

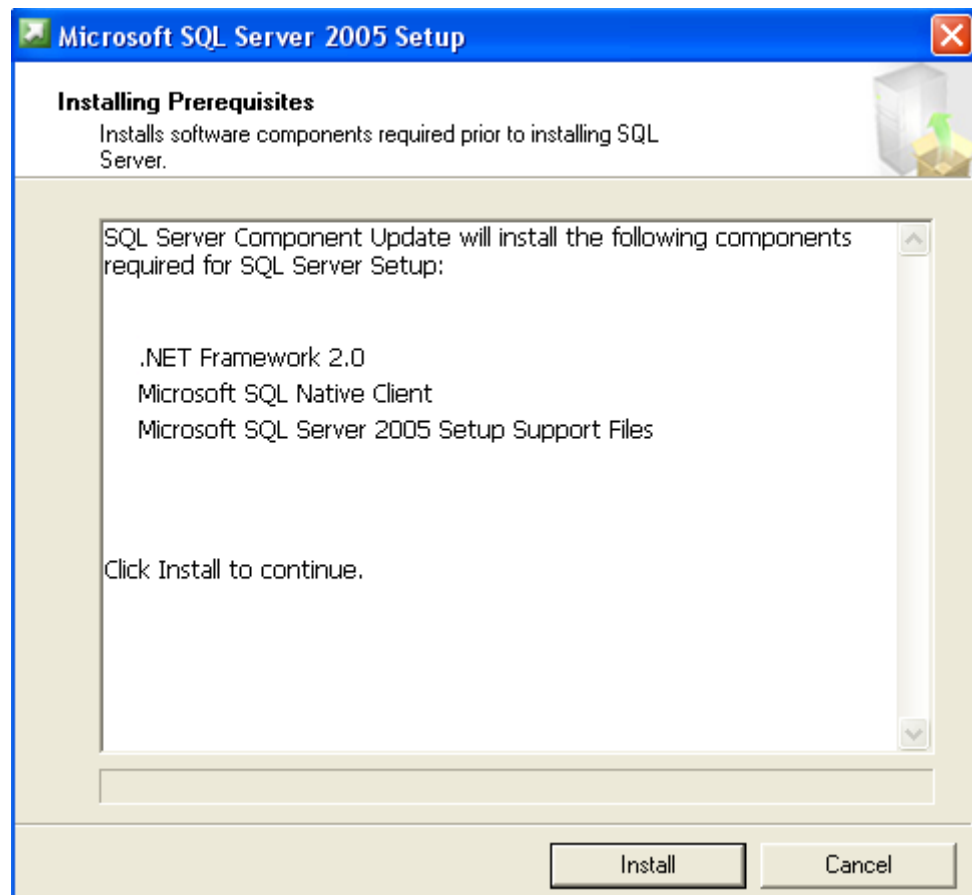


Será mostrada a tela:

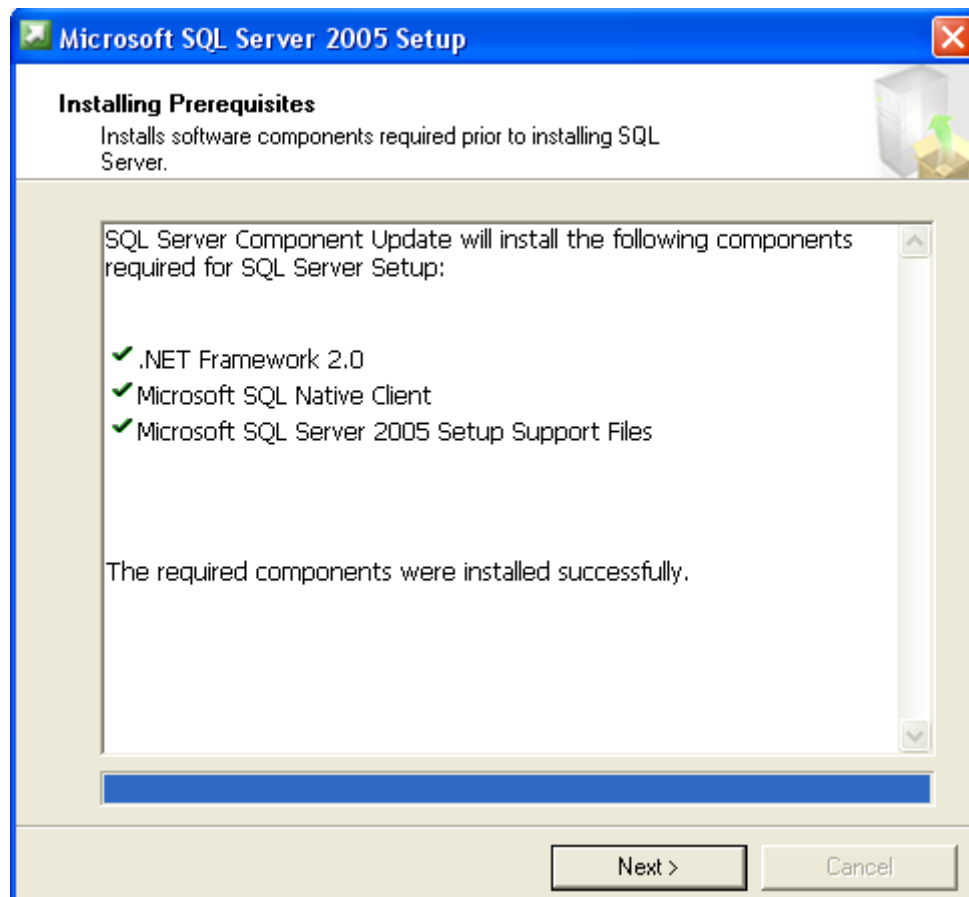


Clique em I accept the licensing terms and conditions e clique em Next >

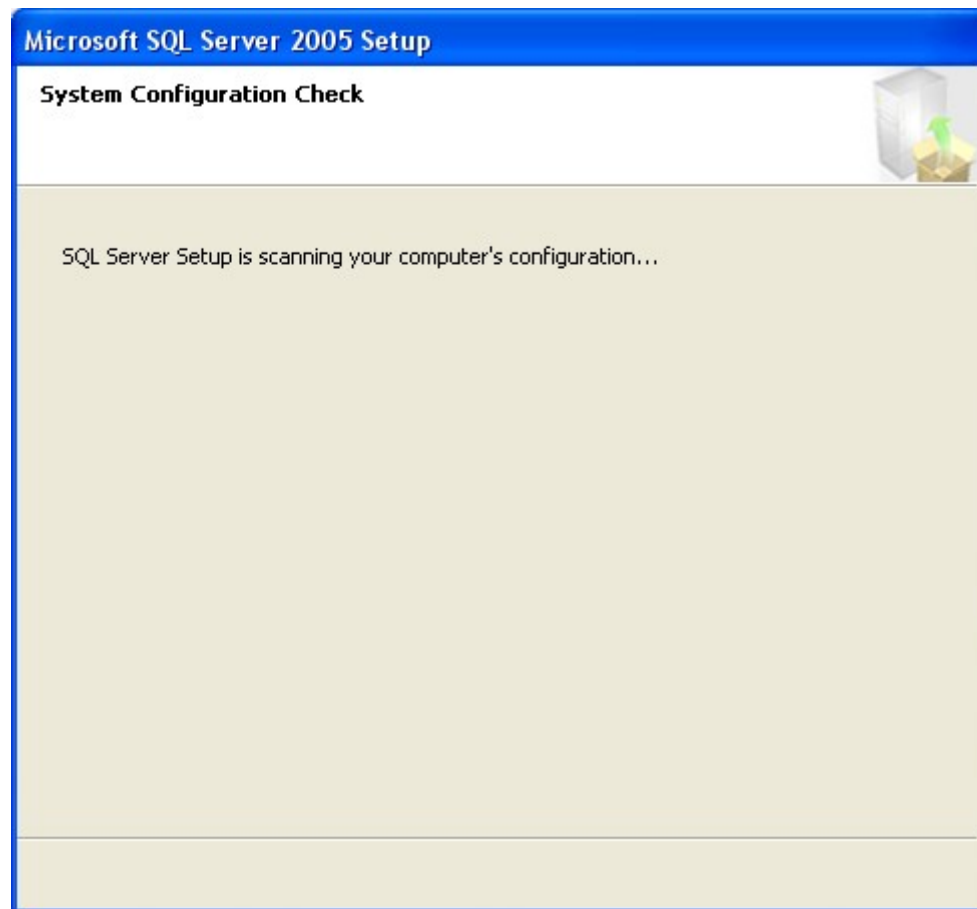
Será mostrada a tela:



Clique em Install



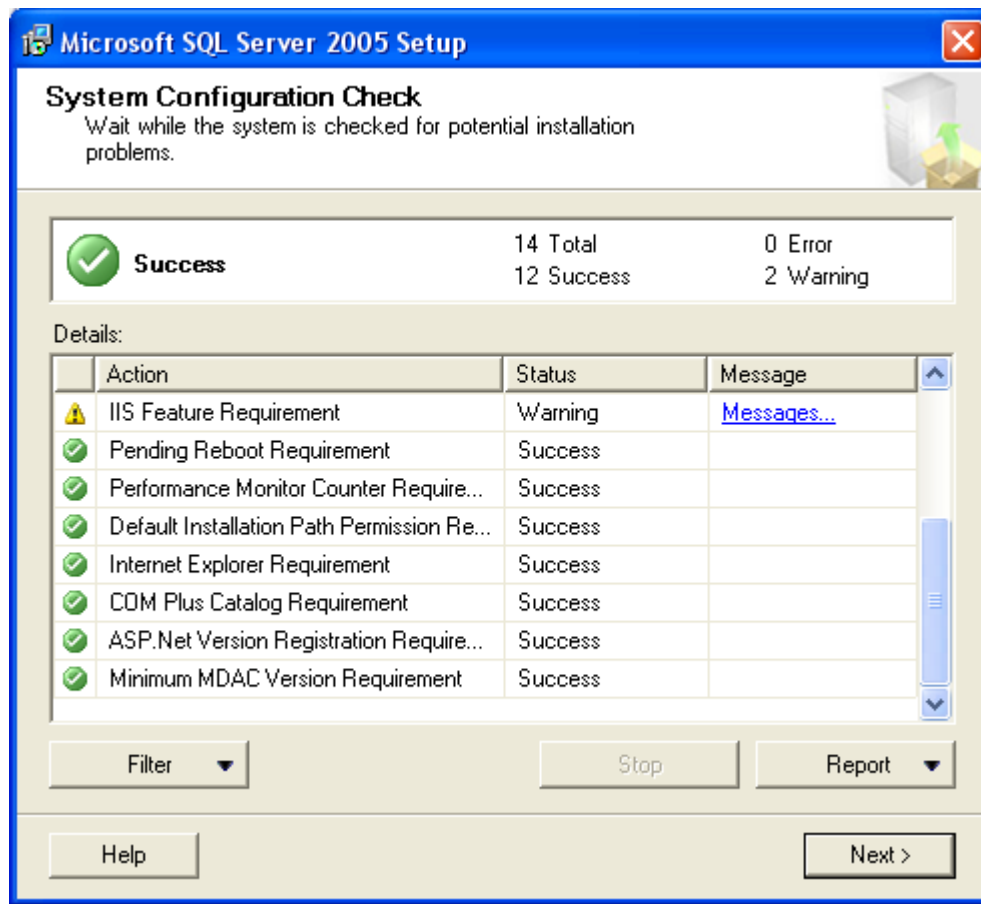
Após instalar componentes necessários para o Microsoft SQLServer 2005, será solicitado clicar em Next > para continuar.



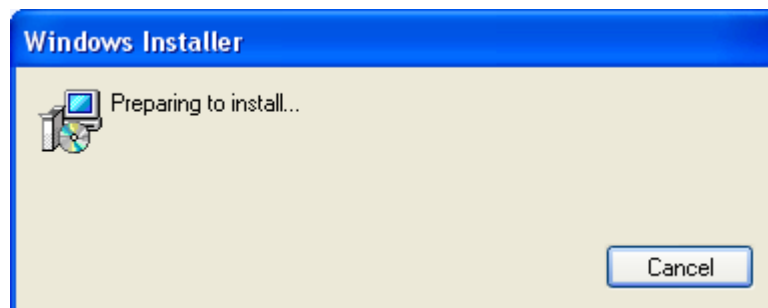
Ir  verificar o computador e



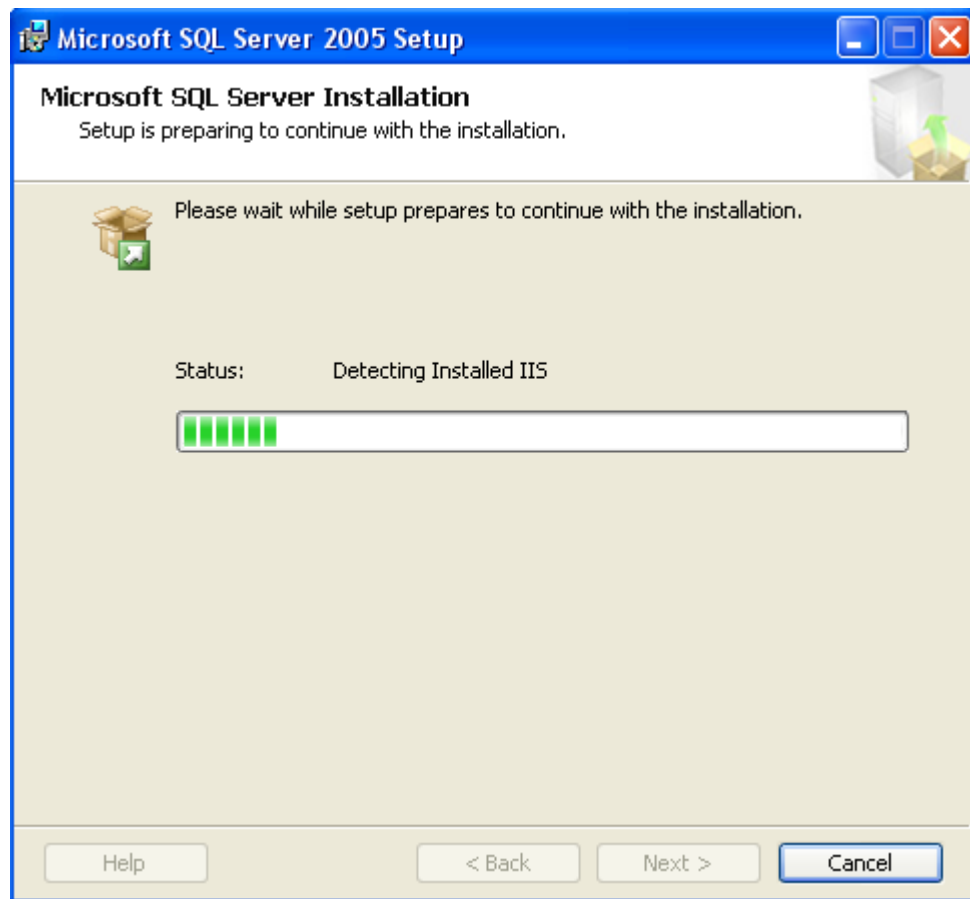
Solicitará que clique em Next > para continuar.



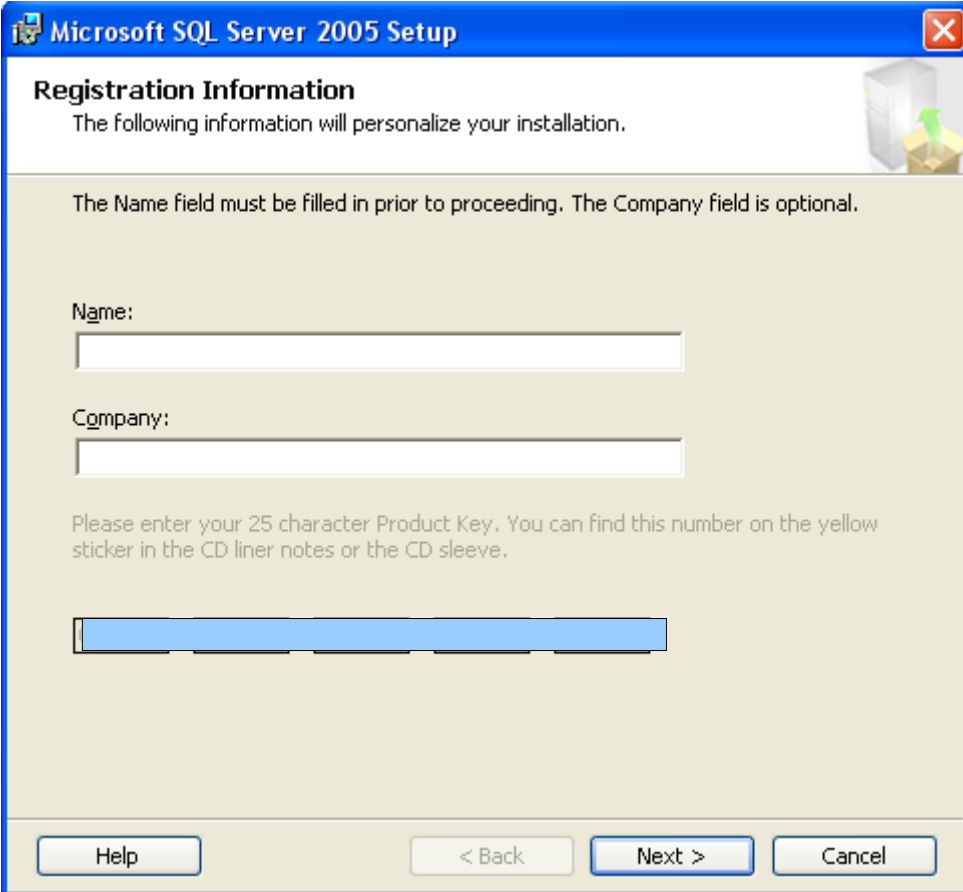
Irá fazer uma nova verificação no sistema operacional e solicitará que clique em Next >



Aguarde um momento e

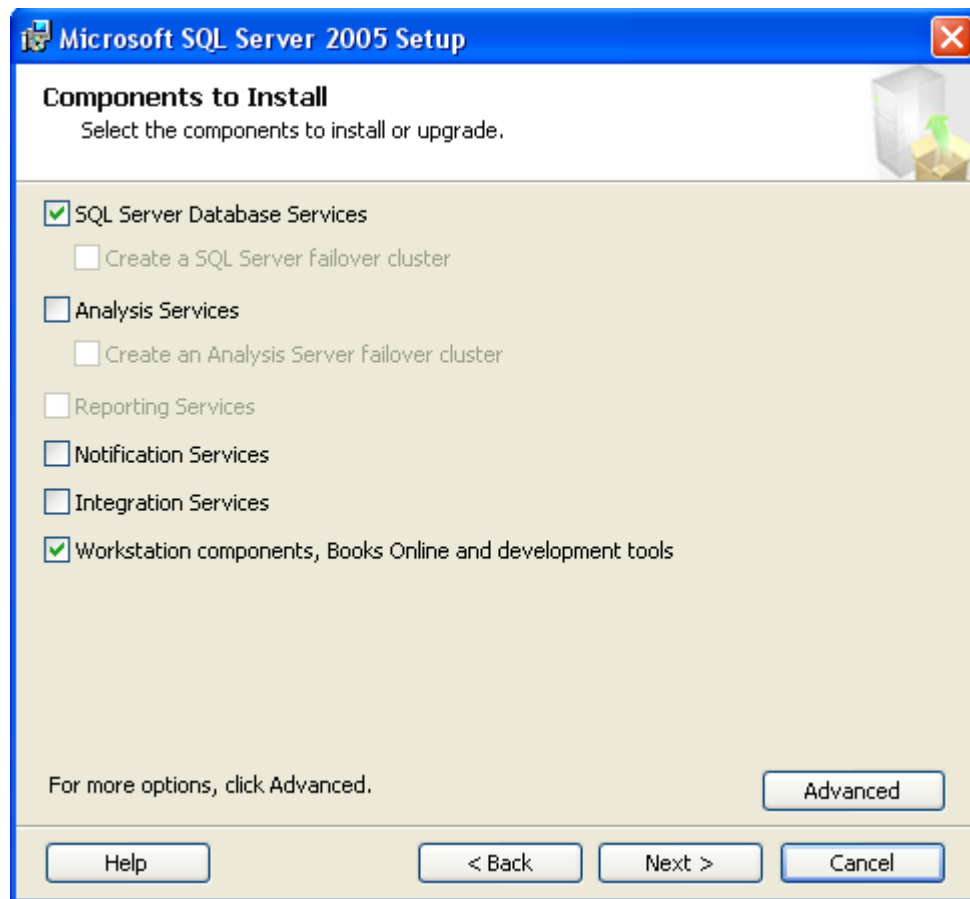


E ...



The screenshot shows the 'Microsoft SQL Server 2005 Setup' window. The title bar is blue with the Microsoft logo and the text 'Microsoft SQL Server 2005 Setup'. The window has a standard Windows XP-style border with a close button (X) in the top right corner. The main content area has a light beige background. At the top, the heading 'Registration Information' is followed by the text 'The following information will personalize your installation.' and a small icon of a server and a box. Below this, a note states: 'The Name field must be filled in prior to proceeding. The Company field is optional.' There are two text input fields: 'Name:' and 'Company:'. Below these is a section for the 'Product Key' with the instruction: 'Please enter your 25 character Product Key. You can find this number on the yellow sticker in the CD liner notes or the CD sleeve.' This is followed by a long, empty text box. At the bottom of the window, there are four buttons: 'Help', '< Back', 'Next >', and 'Cancel'.

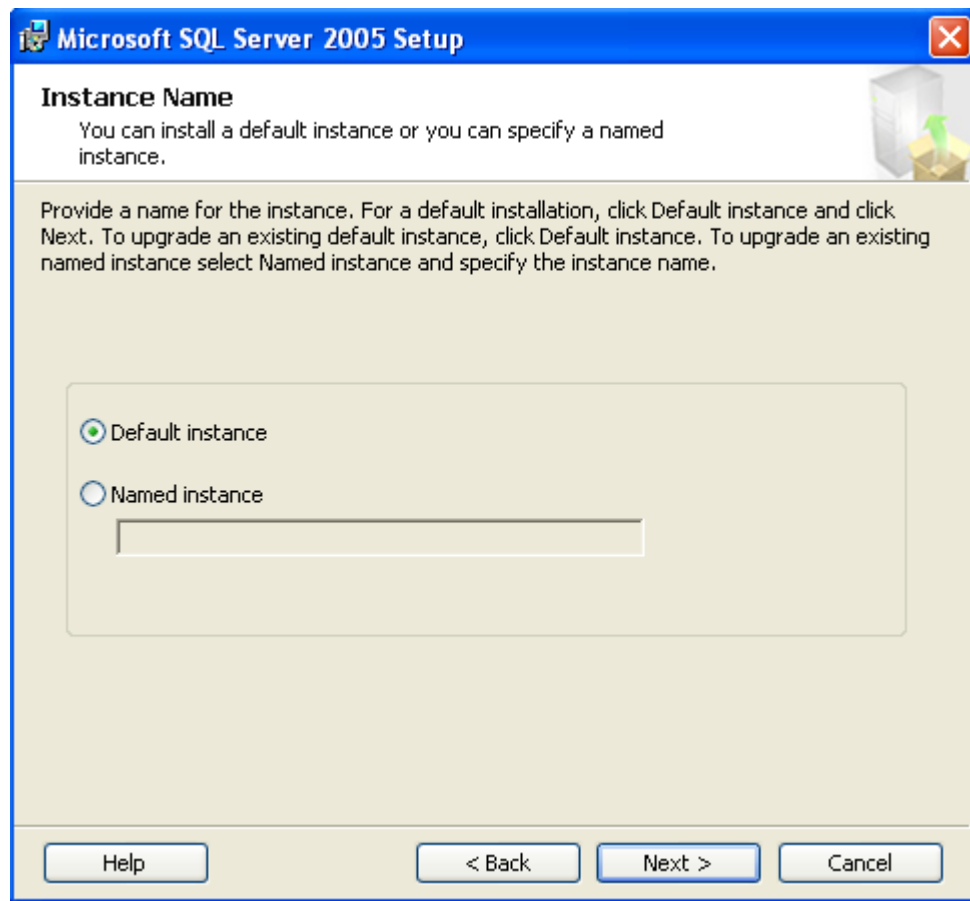
Repare que o serial já estará preenchido (no nosso caso, foi propositalmente escondido o serial). Lembre-se, esta versão é para uso acadêmico, baixado do site da Microsoft, através de acordo entre o Centro Paula Souza e a Microsoft, sendo vetado seu uso comercial. Coloque seu nome e clique em Next >



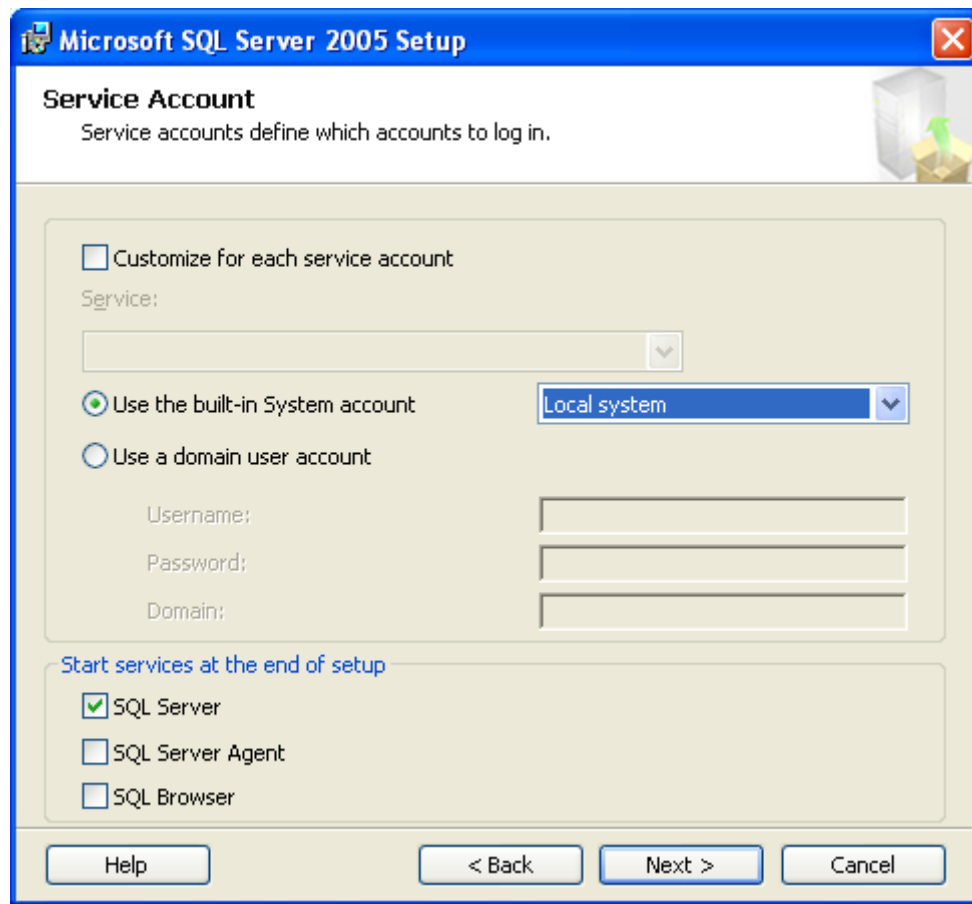
Para o uso escolar, basta selecionar o SQL Server Database Services e o Workstation components, Books Online and development tools.

Caso deseje outros componentes, selecione-os, mas lembre-se: para que os demais componentes funcionem, é necessário que o computador possua o IIS instalado e funcional.

Clique em Next >

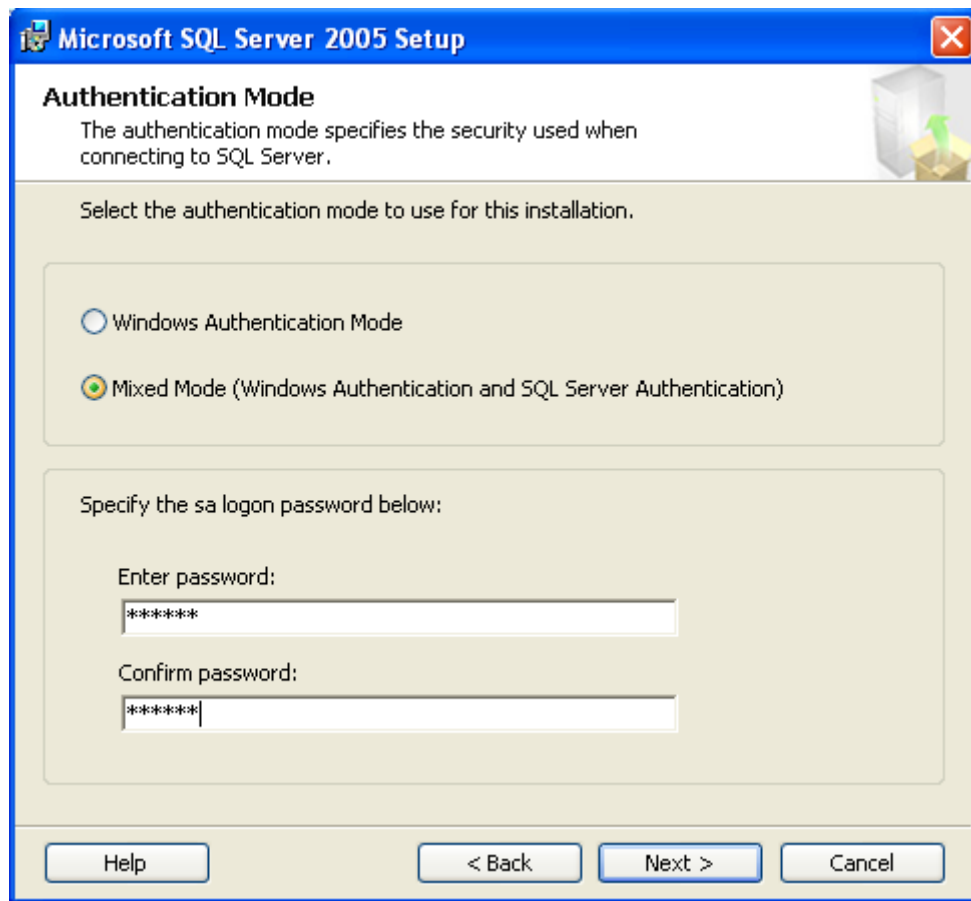


Selecione o Default instance (Instância Padrão) e clique em Next >



Selecione Use the built-in System account e escolha Local system e em Start services at the end of setup, selecione pelo menos SQL Server. Os demais não são necessários para o nosso propósito.

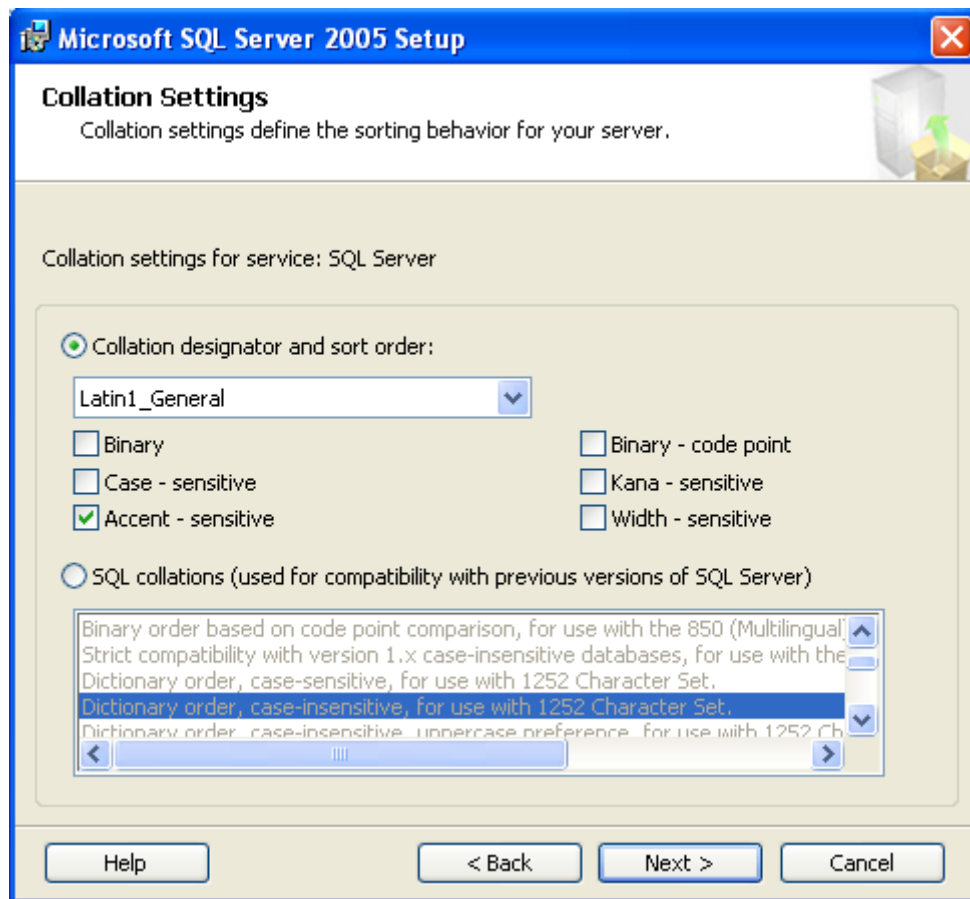
Clique em Next >



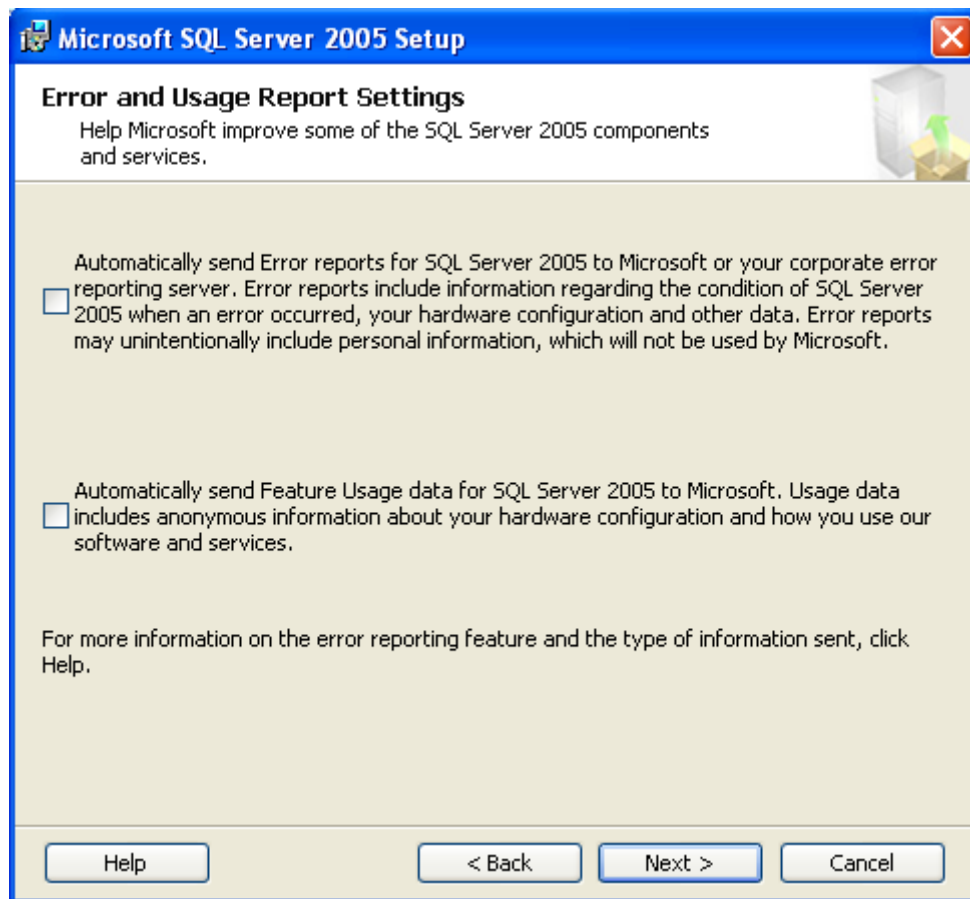
Atenção nesta fase. Selecione Mixed Mode (Windows Authentication and SQL Server Authentication).

Insira uma senha para o Login sa e confirme a senha (no nosso caso, a senha será etecpj). Há distinção entre caixa alta e caixa baixa.

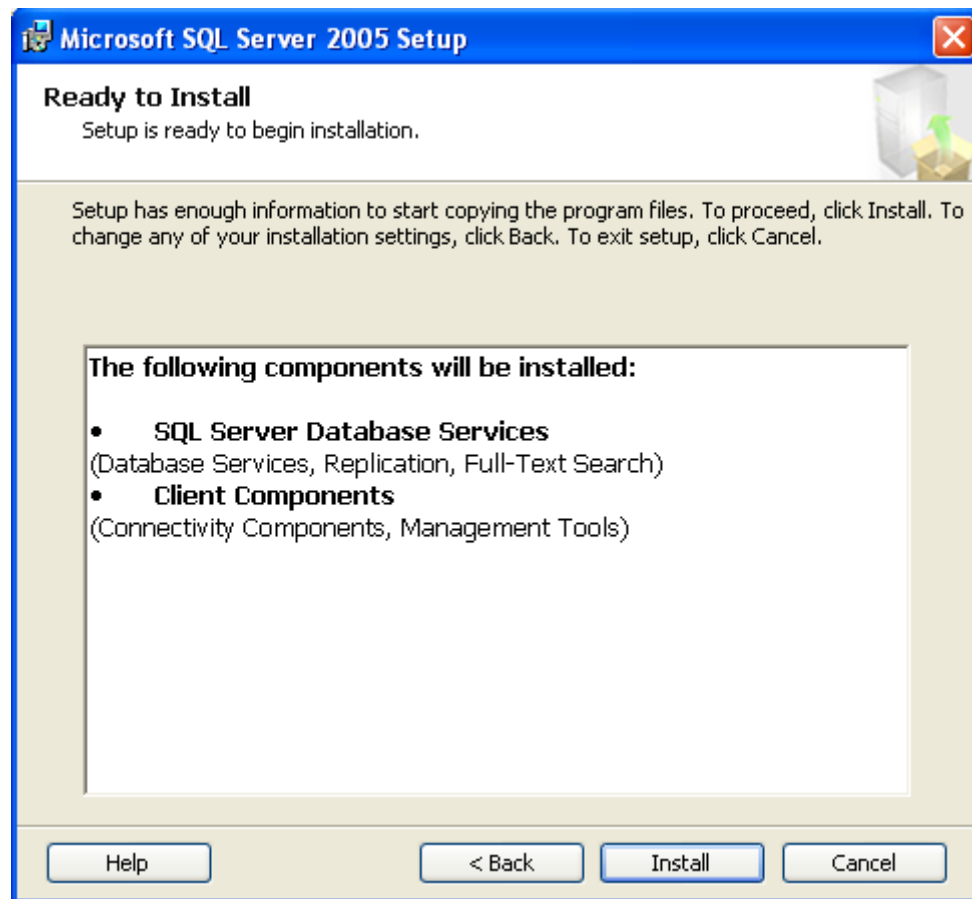
Clique em Next >



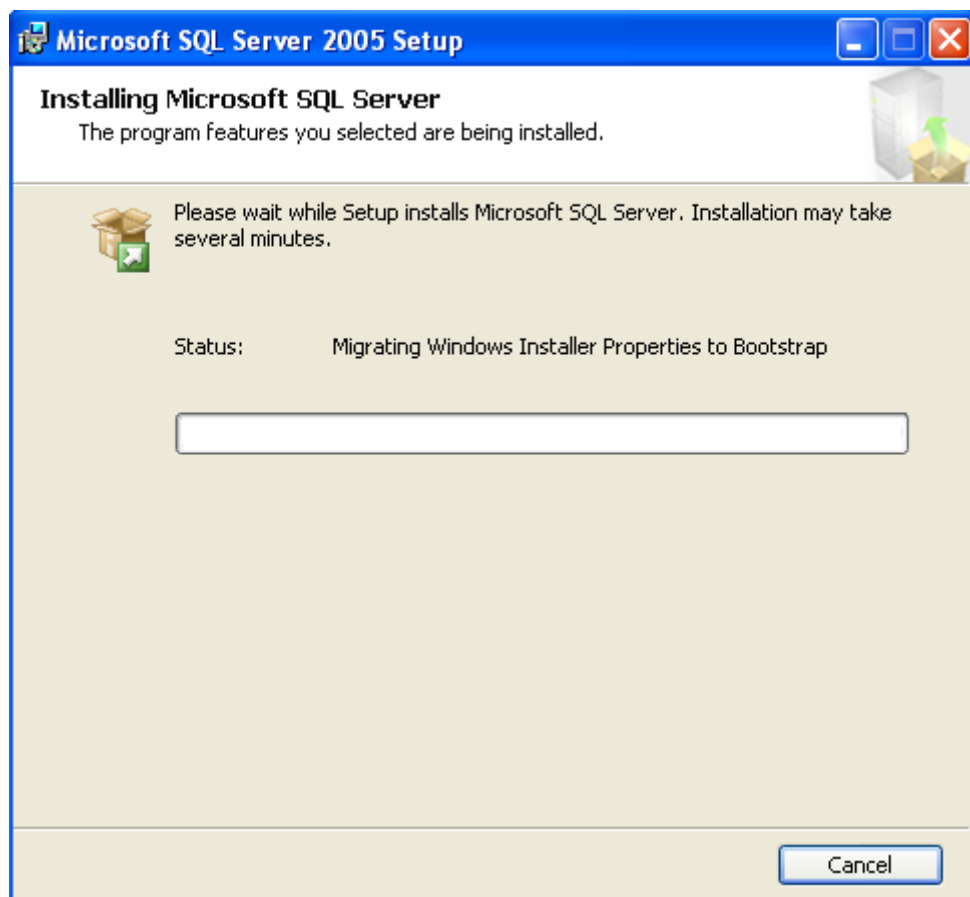
Deixe as opções no padrão e clique em Next >



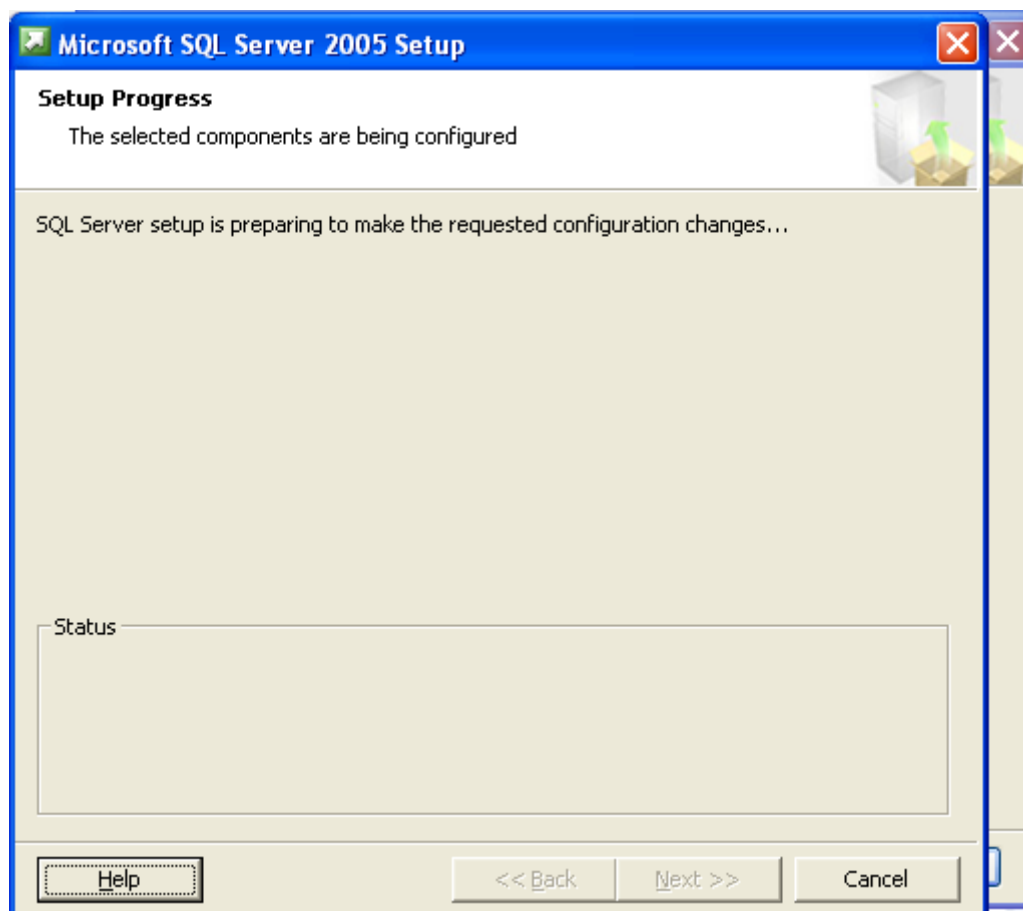
Clique em Next >



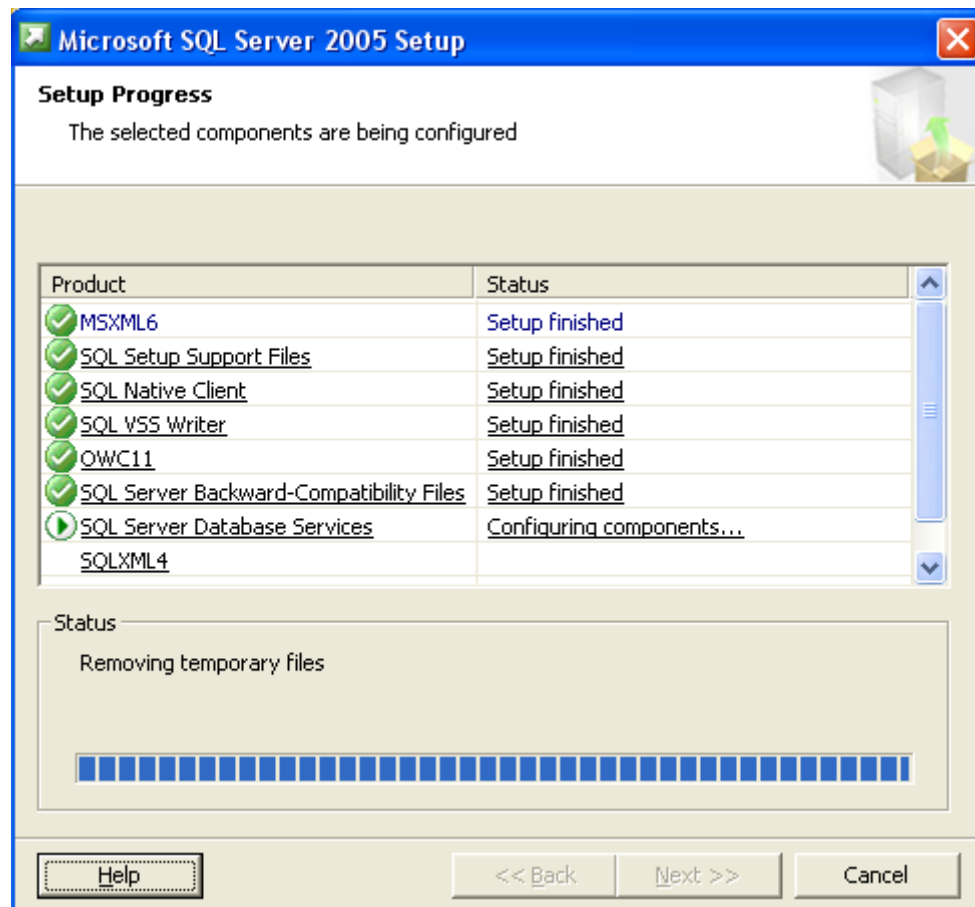
Clique em Install



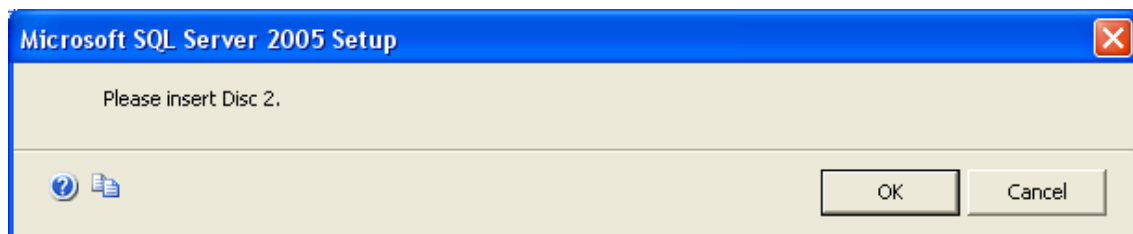
Neste momento, pode ir tomar um café ou fazer um lanche. É demorado mesmo.



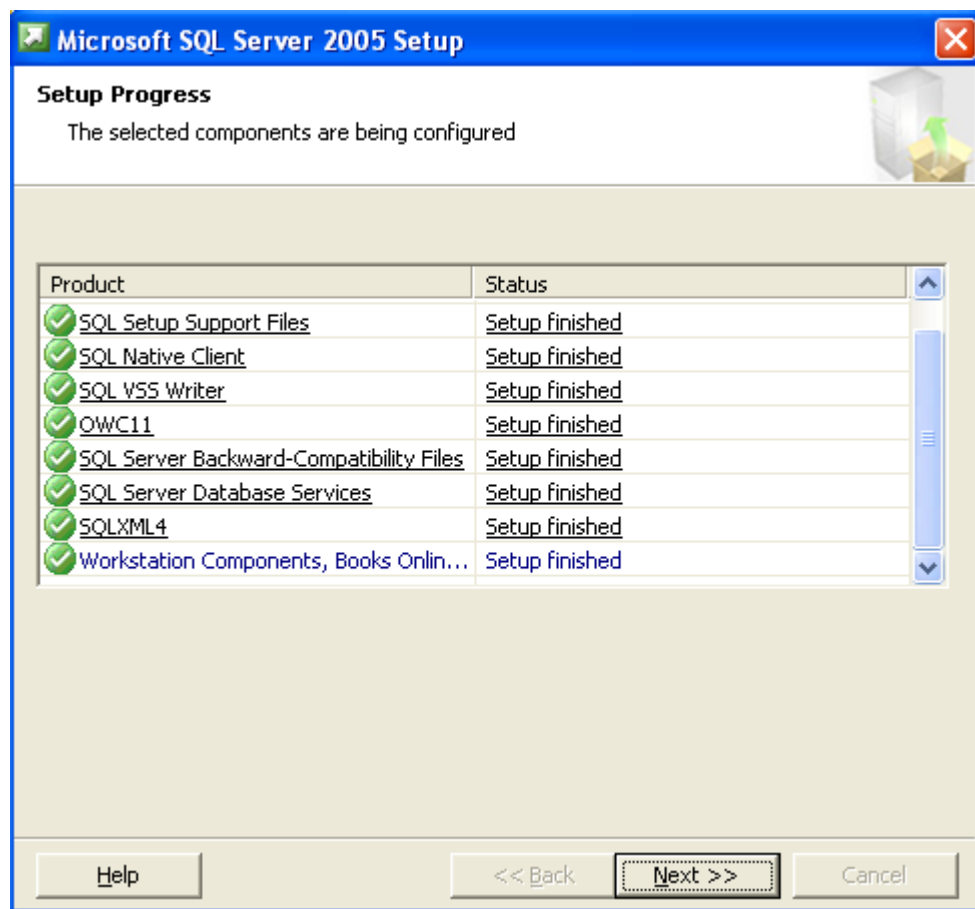
Pode continuar tomando seu café. Manterá você acordado para finalizar a instalação.



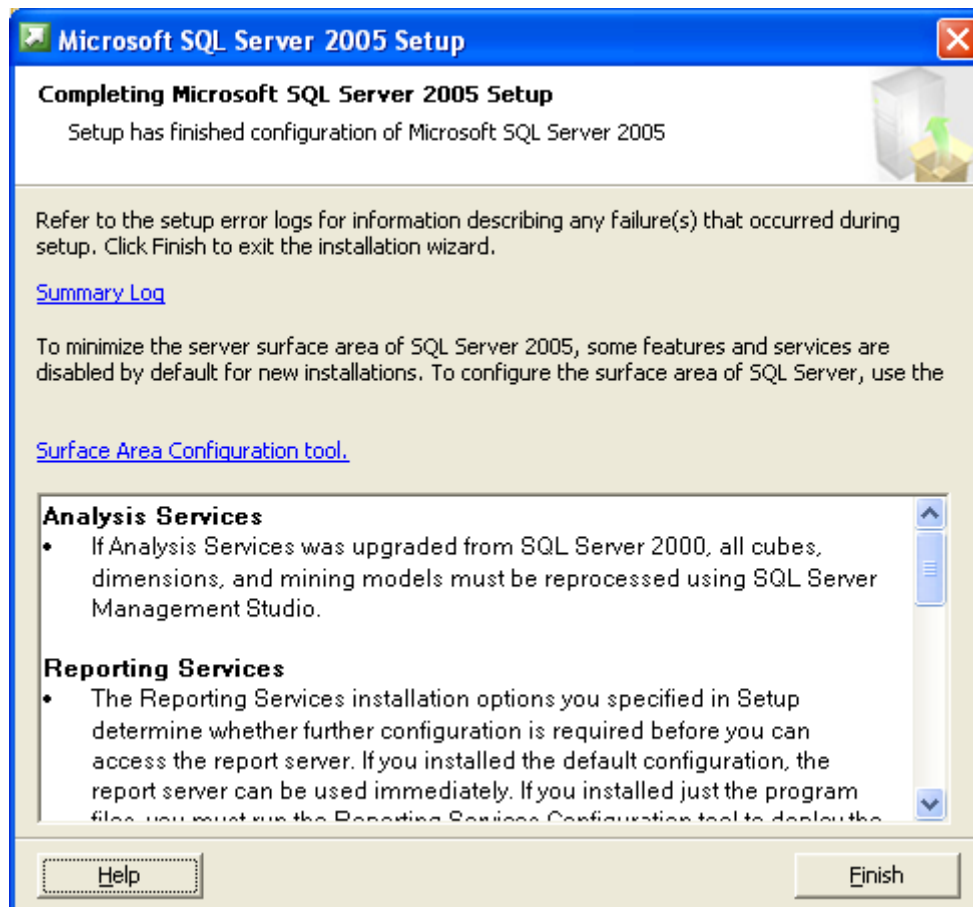
Dá tempo para outro café.



Insira o Disco 2 e clique em Ok. Para quem gosta de café, este é o programa.



Clique em Next >>

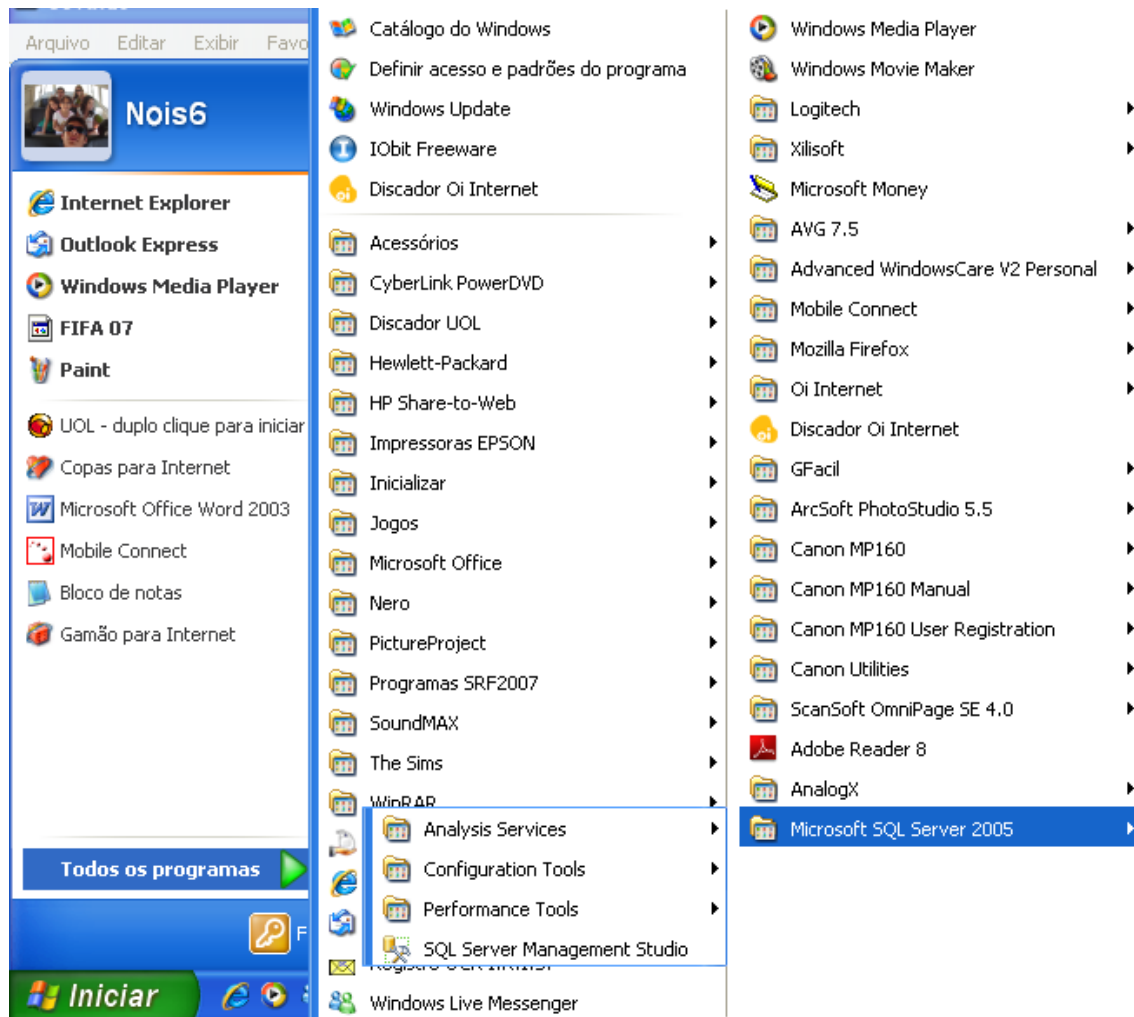


Clique em Finish. A instalação terminou.

Obs:

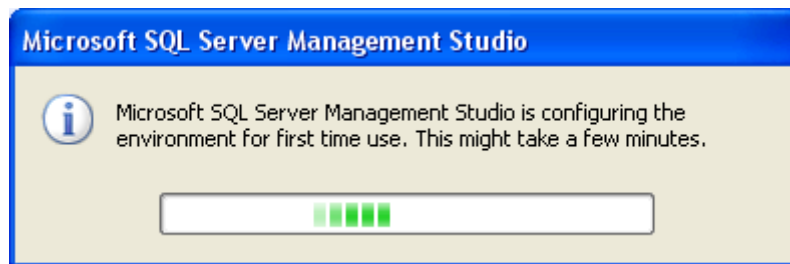
Caso você tenha instalado no seu micro o Visual Studio 2005 ou 2008, aparecerá no início da instalação uma mensagem informando que o:

Workstation components, Books Online and development tools
já está instalado. Desconsidere a informação e MARQUE o Workstation para instalar.

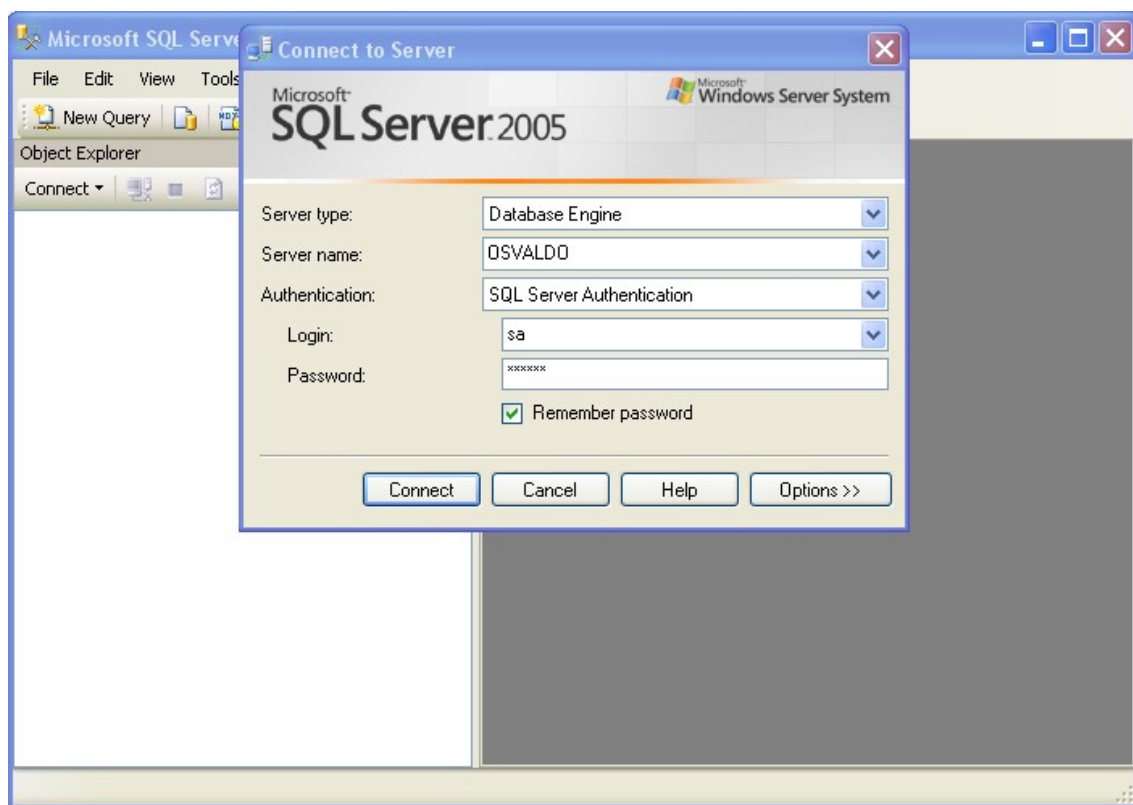


Inicie o SQL Server Management Studio

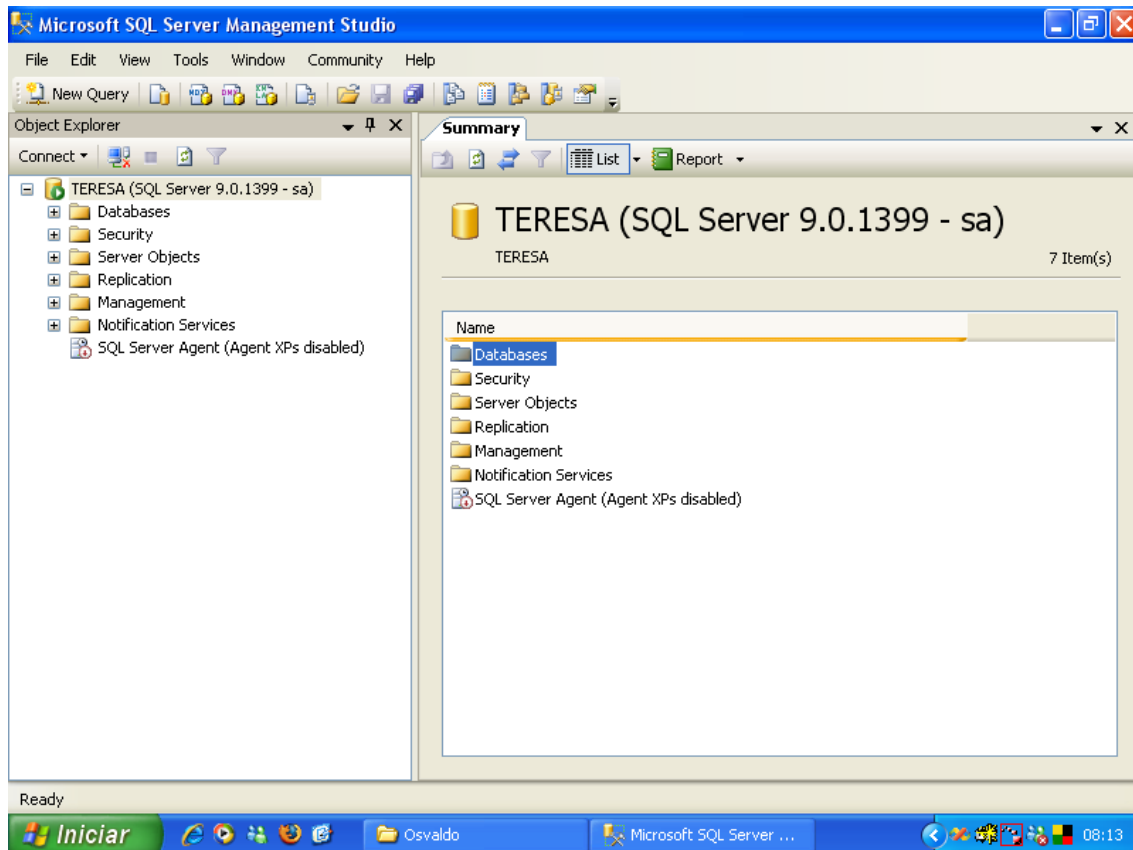




Na primeira vez que iniciar o Microsoft SQL Server Management Studio, será feita uma configuração do sistema. Demora mais ou menos 3 minutos.

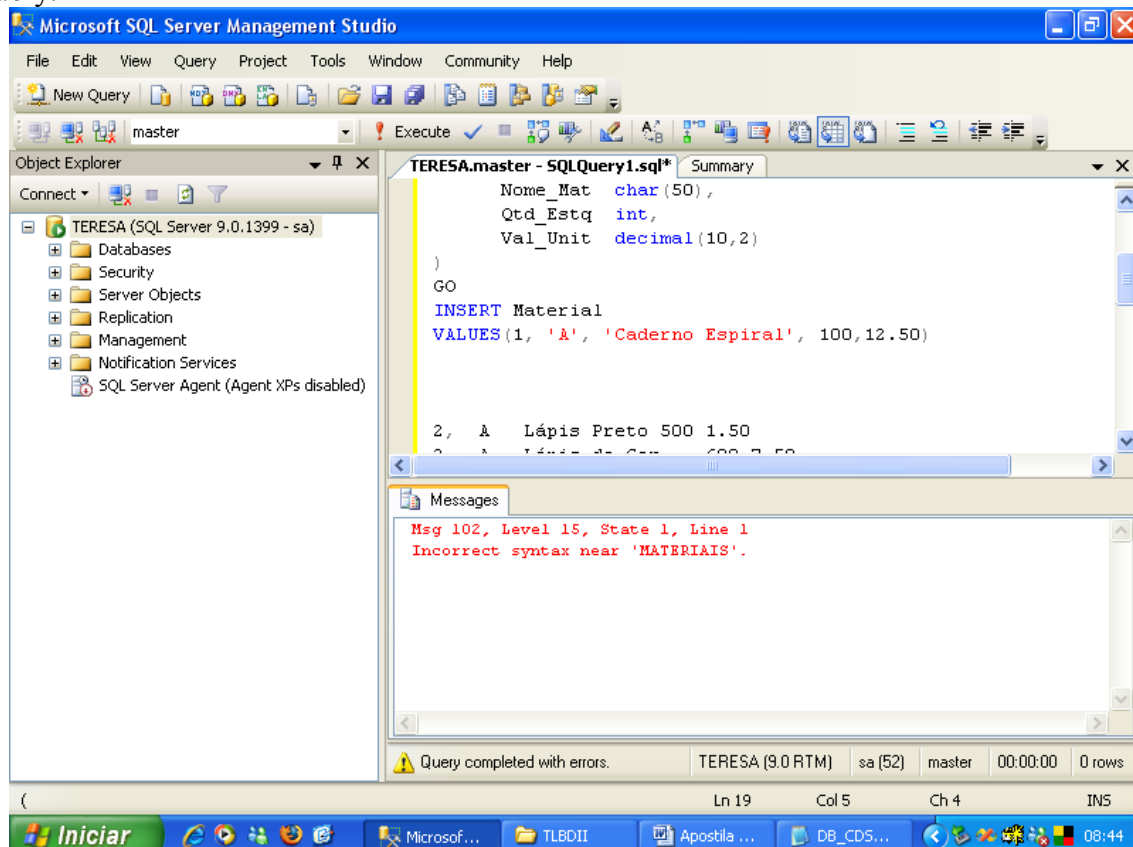


Escolha o modo de autenticação SQL Server Authentication, informe o login as e a senha informada anteriormente. Para que o sistema lembre da senha, selecione Remember password. Clique em Connect



Será mostrada a tela inicial do SQL Server Management.

Para que possa ser inseridos comandos, devemos ter uma tela de console. Clique em New Query.



Trabalhando com o MS SQL SERVER.

1. Sempre será na guia Query
2. Após a digitação do comando, deverá selecionar o comando completo e pressionar a tecla F5
3. Caso deseje que seja numerado as linhas, vá em:

Tools | Options | Text Editor | All Languages | General

e selecione a opção Line Numbers
4. Chamaremos o MS SQL SERVER simplesmente de MSSQL.

O que abordaremos:

- Comandos DDL – Data Definition Language (CREATE, ALTER, DROP);
- Comandos DML – Data Manipulation Language (INSERT, DELETE, UPDATE)
- Comandos DQL – Data Query Language (SELECT)
- Transações
- Procedures;
- Triggers;

O que não abordaremos:

- Comandos DCL – Data Control Language (CREATE USER, ALTER USER, GRANT, REVOKE, CREATE SCHEMA)
- Comandos Avançados

Parte 1

Agora vamos criar nosso primeiro database:

Digite o comando:

```
CREATE DATABASE AULA1
```

Após criado o database, precisamos informar ao MSSQL que queremos utilizá-lo:

Digite o comando:

```
USE AULA1
```

Para sabermos se o MSSQL está usando o database indicado, será indicado em um combobox à sua esquerda o database em uso.

Nota: O MSSQL não distingue caixa alta de caixa baixa (maiúscula de minúscula) internamente, e todos os comandos terminam quando da sua seleção.

1 - COMANDO CREATE TABLE E COMANDO INSERT

Vamos criar nossa primeira tabela:

Digite o comando abaixo:

```
CREATE TABLE CLIENTE  
(  
    COD_CLI    INT,  
    NOME_CLI   CHAR(30)  
)
```

Nota: O comando poderá ser digitado em uma única linha, ou como no exemplo acima. A vantagem de se digitar em várias linhas é que o código fica mais claro e menos poluído.

Vamos agora inserir os dados abaixo:

COD_CLI	NOME_CLI
1	Ana de Souza
2	Mariana de Souza
3	Fabiana de Souza
4	Rosana de Souza
5	Adriana de Souza

Para inserir estes dados, utilizamos o comando INSERT.

```
INSERT INTO CLIENTE (COD_CLI, NOME_CLI)
VALUES (1, 'Ana de Souza')
```

OU

```
INSERT INTO CLIENTE
VALUES (2, 'Mariana de Souza')
```

OU

```
INSERT CLIENTE
VALUES (3, 'Fabiana de Souza')
```

Para visualizar os dados inseridos, digite o comando:

```
SELECT * FROM CLIENTE
```

Vamos agora analisar cada comando:

```
INSERT INTO CLIENTE (COD_CLI, NOME_CLI)
VALUES (1, 'Ana de Souza')
```

Este comando de inserção é do tipo explícito, ou seja, indica qual campo irá receber o dado. Caso fosse colocado o comando:

```
INSERT INTO CLIENTE (NOME_CLI, COD_CLI)
VALUES ('Ana de Souza', 1)
```

O resultado seria o mesmo.

```
INSERT INTO CLIENTE
VALUES (2, 'Mariana de Souza')
```

Este comando de inserção é do tipo posicional. Neste caso, todos os campos da tabela tem que ser informados e na seqüência que foram criados. Caso seja invertido os campos, o MSSQL retornará erro, exceto se os campos forem do mesmo datatype.

```
INSERT CLIENTE
VALUES (3, 'Fabiana de Souza')
```

Neste comando, foi omitido o termo INTO. Este termo é opcional.

Vamos agora excluir a tabela CLIENTE

```
DROP TABLE CLIENTE ←--- comando DROP TABLE
```

Nossa segunda tabela

COD_CLI	NOME_CLI	RUA_CLI	FONE_CLI	EMAIL_CLI
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com
2	Mariana de Souza	Rua B	6524-2541	mariana@hotmail.com
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com
5	Adriana de Souza	Rua E	5263-7451	adriana@hotmail.com

Conectar ao database Aula1:

```
USE AULA1
```

Execute o comando **CREATE TABLE**, como segue:

```
CREATE TABLE CLIENTE
```

```
(
    COD_CLI    INT,
    NOME_CLI   CHAR(30),
    RUA_CLI    CHAR(20),
    FONE_CLI   CHAR(10),
    EMAIL_CLI  CHAR(20)
)
```

Inserir os dados da tabela com o comando INSERT

```
INSERT CLIENTE
```

```
VALUES (1, 'Ana de Souza', 'Rua A',
        '2188-2587', 'ana@hotmail.com')
```

Repetir para os demais registros.

Selecionar todos os registros com **SELECT**.

Elimine a tabela CLIENTE com o comando **DROP TABLE**

Nossa terceira tabela

COD_CLI	NOME_CLI	RUA_CLI	FONE_CLI	EMAIL_CLI	SAL_CLI
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com	1500.23
2	Mariana de Souza	Rua B	6524-2541	mariana@hotmail.com	2800.15
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com	287.56
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com	5589.58
5	Adriana de Souza	Rua E	5263-7451	adriana@hotmail.com	6785.59

O objetivo agora é conhecer o datatype decimal.

Conecte-se ao database Aula1:

```
USE AULA1
```


Execute o comando **CREATE TABLE**:

```
CREATE TABLE CLIENTE
(
    COD_CLI    INT,
    NOME_CLI   CHAR(30),
    RUA_CLI    CHAR(20),
    FONE_CLI   CHAR(10),
    EMAIL_CLI  CHAR(20),
    SAL_CLI    DECIMAL(10,2)
)
```

Inserir os dados da tabela com o comando INSERT

```
INSERT CLIENTE
VALUES (1, 'Ana de Souza', 'Rua A', '2188-2587',
       'ana@hotmail.com', 1500.23)
```

Repetir para os demais registros.

Selecionar todos os registros com **SELECT**.

Elimine a tabela com o comando **DROP TABLE**

OBS:

Para criar uma coluna do tipo decimal, usa-se a vírgula para especificar a parte inteira e a parte decimal.

Para inserirmos um valor neste tipo de coluna, utilizamos o ponto como separador decimal.

Nossa quarta tabela

COD_CLI	NOME_CLI	RUA_CLI	FONE_CLI	EMAIL_CLI	SAL_CLI	DATA_CLI
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com	1500.23	01/15/08
2	Mariana de Souza	Rua B	6524-2541	mariana@hotmail.com	2800.15	02/06/08
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com	287.56	02/24/08
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com	5589.58	02/27/06
5	Adriana de Souza	Rua E	5263-7451	adriana@hotmail.com	6785.59	01/22/08

O objetivo agora é conhecer o datatype varchar e datetime.

Conecte-se ao database Aula1:

```
USE AULA1
```

Execute o comando **CREATE TABLE**:

```
CREATE TABLE CLIENTE
(
    COD_CLI    INT,
    NOME_CLI   CHAR(30),
    RUA_CLI    VARCHAR(20),
    FONE_CLI   CHAR(10),
    EMAIL_CLI  CHAR(20),
    SAL_CLI    DECIMAL(10,2),
    DATA_CLI  DATETIME
)
```

Inserir os dados da tabela com o comando **INSERT**

```
INSERT CLIENTE
VALUES (1, 'Ana de Souza', 'Rua A', '2188-2587',
       'ana@hotmail.com', 1500.23, '01/15/08')
```

Obs: Data é inserida no formato 'MM/DD/AAAA' entre apóstrofos.

Repetir para os demais registros.

Selecionar todos os registros com **SELECT**.

Elimine a tabela com o comando **DROP TABLE**

Explicação sobre os datatypes do MSSQL:

Todos os campos do MSSQL possuem a propriedade datatype. Esta propriedade indica ao MSSQL que tipo de dados serão armazenados. O MSSQL possui os seguintes datatypes:

bigint
binary
bit
char
datetime
decimal
float

image
int
money
nchar
ntext
nvarchar
numeric
real
smalldatetime
smallint
smallmoney
sysname
text
timestamp
tinyint
varbinary
varchar
uniqueidentifier

Obs: Alguns datatypes serão vistos mais adiante. Para maiores detalhes, consultar a documentação do MSSQL.

Nossa quinta tabela:

COD_CLI INT	NOME_CLI CHAR(30)	RUA_CLI VARCHAR(100)	FONE_CLI CHAR(9)	EMAIL_CLI CHAR(30)	SAL_CLI DECIMAL(10,2)	DAT_CLI DATETIME
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com	1500.23	01/15/08
2	Mariana de Souza	Rua B	6524-2541	mariana@hotmail.com	2800.15	02/06/08
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com	287.56	02/24/08
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com	5589.58	02/27/06
5	Adriana de Souza	Rua E	5263-7451	adriana@hotmail.com	6785.59	01/22/08

Conecte-se ao database Aula1:

USE AULA1

Crie a 5ª tabela.

Execute o comando **CREATE TABLE**:

2 - Comando SELECT

Exibir todas as colunas

```
SELECT * FROM CLIENTE
```

Exibir algumas colunas

```
SELECT NOME_CLI, FONE_CLI FROM CLIENTE
```

Exibir algumas colunas utilizando Alias

```
SELECT NOME_CLI AS Nome_Cliente,  
       FONE_CLI AS Telefone_Cliente  
FROM CLIENTE
```

Ou

```
SELECT NOME_CLI Nome_Cliente,  
       FONE_CLI Telefone_Cliente  
FROM CLIENTE
```

Obs 1: Apesar de podermos omitir a clausula **AS**, é preferível que a utilizemos

Obs 2: Alias é um apelido que podemos colocar em uma coluna no momento da exibição de seu valor.

Exibir algumas colunas utilizando alias com espaço no nome:

```
SELECT NOME_CLI AS 'Nome Cliente',  
       FONE_CLI AS 'Telefone Cliente'  
FROM CLIENTE
```

Criando coluna virtual com o comando SELECT

```
SELECT NOME_CLI AS Nome_Cliente,  
       FONE_CLI AS Telefone_Cliente,  
       'Pessoa Física' AS Tipo  
FROM CLIENTE
```

Exibindo um campo calculado com o Comando Select

```
SELECT NOME_CLI AS Nome_Cliente,  
       SAL_CLI AS Telefone_Cliente,  
       SAL_CLI * 1.1 AS 'Salário com 10% de Aumento'  
FROM CLIENTE
```

Exibindo apenas uma vez os dados repetidos – **DISTINCT**

Insere o cliente 5 novamente com código de cliente 6

Se utilizar o comando

```
SELECT NOME_CLI, DATA_CLI FROM CLIENTE
```

Repare que houve repetição de nome.

Tente:

```
SELECT DISTINCT NOME_CLI, DATA_CLI FROM CLIENTE
```

Os dados não repetem.

3 – COMANDO INSERT COM SELECT

INSERT com **SELECT** quando as tabelas são iguais

Crie uma nova tabela com o nome Funcionario com os campos:

COD_FUNC	NOME_FUNC	RUA_FUNC	FONE_FUNC	EMAIL_FUNC	SAL_FUNC	DAT_FUNC
INT	CHAR(30)	VARCHAR(100)	CHAR(9)	CHAR(30)	DECIMAL(10,2)	DATETIME

Digite o comando:

```
INSERT FUNCIONARIO
SELECT * FROM CLIENTE
```

Este comando fará inserção de todos os registros e de todos os campos da tabela CLIENTE na tabela FUNCIONARIO. Para confirmar, digite o comando

```
SELECT * FROM FUNCIONARIO
```

INSERT com **SELECT** quando as tabelas são diferentes.

Exclua a tabela funcionario com o comando **DROP TABLE** e crie uma nova tabela funcionario, com os campos:

COD_FUNC INT	NOME_FUNC CHAR(30)
-----------------	-----------------------

Digite o comando:

```
INSERT INTO FUNCIONARIO
SELECT COD_CLI, NOME_CLI FROM CLIENTE
```

Este comando fará a inserção de todos os registros da tabela cliente na tabela funcionário, porém somente dos campos COD_CLI E NOME_CLI. Para confirmar, digite o comando:

```
SELECT * FROM FUNCIONARIO
```

Exclua a tabela funcionario e crie uma nova tabela funcionario com os campos:

COD_FUNC INT	NOME_FUNC CHAR(30)	RUA_FUNC VARCHAR(100)	FONE_FUNC CHAR(9)	EMAIL_FUNC CHAR(30)	SAL_FUNC DECIMAL(10,2)	CPF_FUNC DATETIME
-----------------	-----------------------	--------------------------	----------------------	------------------------	---------------------------	----------------------

DEPTO_FUNC INT	DAT_FUNC CHAR(30)
-------------------	----------------------

Digite o comando:

```
INSERT INTO FUNCIONARIO
SELECT COD_CLI, NOME_CLI, RUA_CLI, FONE_CLI, EMAIL_CLI,
SAL_CLI, 'SEM CPF', 'SEM DEPTO', DATA_CLI
FROM CLIENTE
```

O comando acima é um **INSERT POSICIONAL**, com a criação de duas colunas virtuais. Execute o comando:

```
SELECT * FROM FUNCIONARIO
```

Agora digite o comando:

```
INSERT FUNCIONARIO(COD_FUNC, NOME_FUNC, RUA_FUNC,  
FONE_FUNC, EMAIL_FUNC, SAL_FUNC, DATA_FUNC)  
SELECT * FROM CLIENTE
```

O comando acima é um **INSERT DECLARATIVO**. Execute o comando:

```
SELECT * FROM FUNCIONARIO
```

Verifique que no primeiro caso, os campos CPF_FUNC e DEPTO_FUNC foram preenchidos. Já no segundo caso, os campos CPF_FUNC E DEPTO_FUNC receberam valores NULL.

Para fixação dos comandos acima, faça os exercícios abaixo:

- 1 – Criar um database chamado Exercicio1
- 2 – Acessar o database Exercicio1
- 3 – No database Exercicio1, criar a tabela chamada Animal apresentada a seguir e inserir os dados adiante.

ANIMAL

COD_ANI	INT	NOM_ANI CHAR(20)
1		Gato
2		Cachorro
3		Papagaio
4		Pássaros
5		Tartaruga
6		Cobra
7		Cavalo
8		Lagarto
9		Elefante
10		Zebra

- 4 – No database Exercicio1, criar a tabela Funcionário e inserir os dados adiante:

FUNCIONARIO

COD_FUNC INT	NOM_FUNC CHAR(30)	SAL_FUNC DECIMAL(10,2)
1	Maria Amélia	1500.23
2	Ângela Pereira	2500.29
3	Sônia Duarte	980.23
4	Paulo Pereira	325.63
5	Ruth Rocha	452.26
6	Raquel Reguete	586.59
7	Sandra Sodré	458.26
8	Tânia Tâmara	875.26
9	Pedro Paulo Pereira	2326.25
10	Geraldo Galhardo	1253.26

5 – No database Exercício1, criar a tabela Professor e inserir os dados adiante:

PROFESSOR

COD_PROF INT	NOM_PROF CHAR(30)	SEX_PROF CHAR(1)	SAL_PROF DECIMAL(10,2)	DAT_PROF DATETIME
1	André Eira	M	2562.23	01/02/98
2	José Pereira	M	2546,25	02/02/99
3	Manoel Beira	M	2584.26	02/03/00
4	Antônio Aroeira	M	1526.58	03/03/00
5	Valéria Zoeira	F	4526.23	03/04/00
6	Vagner Amoreira	M	235.26	04/01/00
7	Valmor Bananeira	M	125.62	05/05/01
8	Arlete Garibeira	F	154.26	06/06/01
9	Joanete Macieira	F	1485.25	07/07/02
10	Claudete Galopeira	F	2587.13	08/08/02

6 – Criar um database chamado Zoologico

7 – Usar o database Zoologico

8 – No database Zoologico, criar a tabela Animal e inserir os dados a seguir:

ANIMAL

COD_ANI INT	TP_ANI CHAR(15)	NOM_ANI CHAR(30)	IDA_ANI TINYINT	VAL_ANI DECIMAL(10,2)
1	Cachorro	Djudi	3	300.00
2	Cachorro	Sula	5	300.00
3	Cachorro	Sarina	7	300.00
4	Gato	Pipa	2	500.00
5	Gato	Sarangue	2	500.00
6	Gato	Clarences	1	500.00
7	Coruja	Agnes	0	700.00
8	Coruja	Arabela	1	700.00
9	Sapo	Quash	1	100.00
10	Peixe	Fish	0	100.00

9 – Escrever um comando que exiba todas as colunas e todas as linhas da tabela Animal:

10 – Escrever um comando que exiba apenas as colunas TP_ANI, NOM_ANI E IDA_ANI da tabela Animal, apresentando todos os registros da tabela:

11 – Escrever um comando que exiba apenas as colunas TP_ANI E NOM_ANI da tabela animal, apresentando todos os registros da tabela. Apresentar a coluna TP_ANI com o alias Tipo, e a coluna NOM_ANI com a alias Nome;

12 – Escrever um comando que exiba apenas as colunas TP_ANI, NOM_ANI E IDA_ANI da tabela Animal, apresentando todos os registros da tabela. Apresentar a coluna TP_ANI com a alias Tipo, a coluna NOM_ANI com o alias Nome e a coluna IDA_ANI com o alias Tempo de vida;

13 – Escrever um comando que apresente os dados da tabela Animal da seguinte maneira:

Procedência	Tipo	Nome	Tempo de Vida
Animal Doméstico	Cachorro	Djudi	3
Animal Doméstico	Cachorro	Sula	5
Animal Doméstico	Cachorro	Sarina	7
Animal Doméstico	Gato	Pipa	2
Animal Doméstico	Gato	Sarangue	2
Animal Doméstico	Gato	Clarences	1
Animal Doméstico	Coruja	Agnes	0
Animal Doméstico	Coruja	Arabela	1
Animal Doméstico	Sapo	Quash	1
Animal Doméstico	Peixe	Fish	0

Nota: Tempo de Vida é um alias para a coluna IDA_ANI

14 – Escrever um comando que apresente os dados da tabela Animal da seguinte maneira:

Tipo	Nome	Idade	Valor Original	Valor de Venda
Cachorro	Djudi	3	300.00	330.00
Cachorro	Sula	5	300.00	330.00
Cachorro	Sarina	7	300.00	330.00
Gato	Pipa	2	500.00	550.00
Gato	Sarangue	2	500.00	550.00
Gato	Clarences	1	500.00	550.00
Coruja	Agnes	0	700.00	770.00
Coruja	Arabela	1	700.00	770.00
Sapo	Quash	1	100.00	110.00
Peixe	Fish	0	100.00	110.00

Nota: Valor de Venda é um alias referente à coluna virtual que apresenta o resultado da seguinte multiplicação: VAL_ANI * 1.1

15 – Escrever um comando que apresente os dados da tabela Animal da seguinte maneira:

Tipo	Valor Original	Valor de Venda
Cachorro	300.00	330.00
Cachorro	300.00	330.00
Cachorro	300.00	330.00
Gato	500.00	550.00
Gato	500.00	550.00
Gato	500.00	550.00
Coruja	700.00	770.00
Coruja	700.00	770.00
Sapo	100.00	110.00
Peixe	100.00	110.00

Nota: Valor de Venda é um alias referente à coluna virtual que apresenta o resultado da seguinte multiplicação: VAL_ANI * 1.1

16 – Escrever um comando que apresente os dados da tabela Animal da seguinte maneira, apresentando apenas uma vez os dados repetidos:

Tipo	Valor Original	Valor de Venda
Cachorro	300.00	330.00
Gato	500.00	550.00
Coruja	700.00	770.00
Sapo	100.00	110.00
Peixe	100.00	110.00

Nota: Valor de Venda é um alias referente à coluna virtual que apresenta o resultado da seguinte multiplicação: VAL_ANI * 1.1

17 – Criar o database PESSOA

18 – Usar o database PESSOA

19 – No database PESSOA, criar a tabela DOCE seguinte e inserir os dados na seguinte seqüência: NOM_DOC, QTD_DOC, COD_DOC, VAL_DOC:

DOCE

COD_DOC INT	NOM_DOC CHAR(30)	QTD_DOC INT	VAL_DOC DECIMAL(10,2)
1	Brigadeiro	10	30.00
2	Pé-de-Moleque	5	25.00
3	Beijinho	25	30.00
4	Goiabada	1	5.00
5	Olho de Sogra	3	15.00
6	Cocada	2	15.00
7	Doce de Leite	10	20.00
8	Suspiro	5	15.00
9	Doce de Batata Doce	4	5.00
10	Marmelada	2	5.00

20 – No database PESSOAS, criar a tabela CRIANCA a seguir e inserir os dados na seguinte seqüência: SERIE_CRI, SEX_CRI, NOM_CRI, COD_CRI;

CRIANCA

COD_CRI INT	NOM_CRI CHAR(30)	SEX_CRI CHAR(1)	SERIE_CRI TINYINT
1	Rosinha	F	1
2	Ritinha	F	2
3	Margaridinha	F	1
4	Marquinhos	M	2
5	Claudio	M	1
6	Paulino	M	2
7	Carlinhos	M	3
8	Pedrinho	M	3
9	Aninha	F	3
10	Paulinha	F	3

21 – No database PESSOA, criar a tabela PESSOA a seguir:

PESSOA

COD_PES INT	NOM_PES CHAR(30)	SEX_PES CHAR(1)	SERIE_PES TINYINT
----------------	---------------------	--------------------	----------------------

22- Copiar os dados da tabela CRIANCA para a tabela PESSOA;

23 – No database PESSOA, criar a tabela COLEGA a seguir:

COLEGA

NOM_COL CHAR(30)	SEX_COL CHAR(1)
---------------------	--------------------

24 – Copiar os dados da tabela CRIANCA para a tabela COLEGA;

25 – No database PESSOA, criar a tabela GENTE a seguir:

GENTE

COD_GEN INT	NOM_GEN CHAR(30)	SEX_GEN CHAR(1)	SER_GEN TINYINT	DT_GEN DATETIME	TEL_GEN CHAR(10)
----------------	---------------------	--------------------	--------------------	--------------------	---------------------

26 – Copiar os dados da tabela CRIANCA PARA A TABELA GENTE.

Parte II

1 – Comando **Update**

É o comando utilizado para alterar dados de uma tabela. Para que possamos

conhecer as formas de alteração de dados em uma tabela, utilizaremos o exemplo da tabela **Cliente** apresentada a seguir:

Cliente

COD_CLI	NOME_CLI	RUA_CLI	FONE_CLI	EMAIL_CLI	SAL_CLI	DATA_CLI
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com	1500.23	08/15/01
2	Mariana de Souza	Rua B	6224-2587	mariana@hotmail.com	2800.15	06/06/02
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com	287.56	06/24/02
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com	5589.58	06/27/02
5	Adriana de Souza	Rua E	5663-7451	adriana@hotmail.com	6785.59	06/22/01
6	Adriana de Souza	Rua F	3125-4152	drica@hotmail.com	2750.00	06/22/01
7	Cristiana de Souza	Rua G	2541-1542	cristiana@hotmail.com	1650.00	03/03/03
8	Emiliana de Souza	Rua H	9187-6589	emiliana@hotmail.com	1650.00	03/03/03

Alterando dados de uma coluna

Para alterar dados de apenas uma coluna da tabela **Cliente**, executamos o comando seguinte:

```
UPDATE Cliente  
SET Sal_Cli = Sal_Cli * 1.1
```

Esse comando aumenta em 10% o salário de todos os **Clientes** e grava a alteração na tabela.

Após a alteração dos dados, a tabela devera ficar desta forma:

Cliente

COD_CLI	NOME_CLI	RUA_CLI	FONE_CLI	EMAIL_CLI	SAL_CLI	DATA_CLI
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com	1650.25	08/15/01
2	Mariana de Souza	Rua B	6224-2587	mariana@hotmail.com	3180.165	06/06/02
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com	316.316	06/24/02
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com	6148.538	06/27/02
5	Adriana de Souza	Rua E	5663-7451	adriana@hotmail.com	7464.149	06/22/01
6	Adriana de Souza	Rua F	3125-4152	drica@hotmail.com	3025.00	06/22/01
7	Cristiana de Souza	Rua G	2541-1542	cristiana@hotmail.com	1815.00	03/03/03
8	Emiliana de Souza	Rua H	9187-6589	emiliana@hotmail.com	1815.00	03/03/03

Alterando dados de várias colunas

Para alterar dados de varias colunas da tabela Cliente, executamos o comando seguinte:

```
UPDATE Cliente
SET Sal_Cli = Sal_Cli * 1,1,
    Data_Cli = Data_Cli + 1
```

Esse comando aumenta em 10% o salário de todos os Clientes, acrescenta um dia na data de cadastro dos mesmos e grava a alteração na tabela.

Após a alteração dos dados, a tabela devera ficar desta forma:

Cliente

COD_CLI	NOME_CLI	RUA_CLI	FONE_CLI	EMAIL_CLI	SAL_CLI	DATA_CLI
1	Ana de Souza	Rua A	2188-2587	ana@hotmail.com	1815.275	08/16/01
2	Mariana de Souza	Rua B	6224-2587	mariana@hotmail.com	3498.181	06/07/02
3	Fabiana de Souza	Rua C	4578-5689	fabiana@hotmail.com	347.947	06/25/02
4	Rosana de Souza	Rua D	3212-3212	rosana@hotmail.com	6763.391	06/28/02
5	Adriana de Souza	Rua E	5663-7451	adriana@hotmail.com	8210.563	06/23/01
6	Adriana de Souza	Rua F	3125-4152	drica@hotmail.com	3327.50	06/23/01
7	Cristiana de Souza	Rua G	2541-1542	cristiana@hotmail.com	1996.50	03/04/03
8	Emiliana de Souza	Rua H	9187-6589	emiliana@hotmail.com	1996.50	03/04/03

2 – Comando Delete

Para excluir os dados de uma tabela, executamos o comando **DELETE**.

Observemos o exemplo a seguir:

```
DELETE FROM Cliente
```

3 – A cláusula WHERE

Introdução

A cláusula **WHERE** é utilizada pelos comandos **SELECT**, **UPDATE** e **DELETE**.

Ela serve para filtrar os dados que serão afetados por esses comandos. Por exemplo, suponhamos que o usuário tenha uma tabela de produtos (como mostra a tabela a seguir) e que ele queira aplicar um aumento de 10% em todos os produtos do tipo '**Sala**'.

Nesse caso, ele teria que escrever um comando que aplicasse esse aumento de valor para alguns produtos e não para todos. Sendo assim, é

necessário utilizar uma cláusula que filtre os dados a serem modificados pela alteração de valor. Essa cláusula chama-se **WHERE**.

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1500.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	200	1200.00
4	Cama	Quarto	300	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	250.00
8	Estante	Sala	800	400.00
9	Penteadeira	Quarto	200	300.00
10	Cadeira	Cozinha	1000	150.00

```
CREATE DATABASE PRODUTOS
GO
USE PRODUTOS
GO
CREATE TABLE Produto
(
    Cod_Prod int,
    Nome_Prod char(50),
    Tipo_Prod char(20)
    Qtd_Prod int,
    Val_Prod decimal(10,2)
)
GO
INSERT Produto
VALUES(1, 'Geladeira', 'Cozinha', 500, 1500.00)
```

Insira os demais registros.

O comando para realizar a alteração de preços nos produtos do tipo **Sala** é:

```
UPDATE Produto
SET Val_Prod = Val_Prod * 1.1
WHERE Tipo.Prod = 'Sala'
```

Depois, se quiséssemos exibir apenas os produtos do tipo **'Sala'**, bastaria utilizar o comando **SELECT** também com a cláusula **WHERE**.

Observemos:

```
SELECT * FROM Produto WHERE Tipo_Prod = 'Sala'
```

Repare que os valores da mercadoria foram atualizados.

Suponhamos agora que o usuário quisesse aumentar a quantidade em estoque de todos os produtos cuja quantidade em estoque seja igual ou inferior a 300 unidades em mais 100 unidades.

Para tanto, deveríamos escrever este comando;

```
UPDATE Produto
  SET Qtd_Prod = Qtd_Prod + 100
  WHERE Qtd_Prod <= 300
```

3.1 - A cláusula WHERE com os operadores AND e OR

Os operadores **AND** e **OR** são chamados de operadores lógicos, e são utilizados quando se pretende realizar mais de uma condição de comparação na cláusula **WHERE**. Por exemplo, suponhamos que, na tabela **Produto** anterior, quiséssemos aplicar um aumento de 10% em todos os produtos de Sala que estão com o preço inferior a 1000.00.

Para executar essa tarefa, temos que utilizar duas condições:

- Produtos do tipo 'Sala';
- Preço inferior a 1000.00.

Nesse caso, as duas condições têm que ser atendidas para que o produto ganhe o aumento de valor. Sendo assim, será necessária a utilização do operador **AND** na cláusula WHERE.

Observemos o comando:

```
UPDATE Produto SET Val_Prod = ValProd * 1.1
  WHERE Tipo_Prod = 'Sala'
        AND Val_Prod < 1000.00
```

Ao executar o comando SELECT que segue, obtemos como resposta os dados seguintes:

```
SELECT * FROM Produto
```


Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1500.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1320.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	302.50
8	Estante	Sala	800	484.00
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

Observemos que apenas os produtos de código 7 e 8 tiveram seu preço alterado.

Se, por outro lado, quiséssemos aplicar um aumento de 10% no valor de todos os produtos que são do tipo Sala, ou que estejam com o preço inferior a 1000.00, teríamos que utilizar o operador **OR**.

Para executar essa tarefa, temos que lançar mão de duas condições:

- Produtos do tipo 'Sala' ou
- Preço inferior a 1000.00.

Nesse caso, as duas condições têm que ser atendidas para que o produto ganhe o acréscimo desejado. Sendo assim, será necessária a utilização do operador **OR** na cláusula **WHERE**.

Observe o comando:

```
UPDATE Produto
  SET Val_Prod = Val_Prod * 1.1
  WHERE Tipo_Prod = 'Sala'
        OR Val_Prod < 1000.00
```

Ao executarmos o comando SELECT que segue, obtemos como resposta estes dados:

```
SELECT * FROM Produto
```

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

Observemos que, com o operador **OR**, muito mais linhas da tabela foram atingidas pelo comando **UPDATE**.

Por causa disso que:

- O operador **OR** é muito mais abrangente do que o operador **AND**;
- O operador **AND** é muito mais restritivo do que o operador **OR**.

3.2 – A cláusula **WHERE** com o operador **IN**

O operador **IN** muitas vezes poderá ser utilizado no lugar do operador **OR**.

Suponhamos que o usuário precisasse selecionar todos os produtos de sala ou de quarto. Para tanto, poderíamos utilizar o operador **OR** ou o operador **IN**.

Vejamos:

Usando o operador **OR**:

```
SELECT * FROM Produto
WHERE Tipo_Prod = 'Sala' OR Tipo_Prod = 'Quarto'
```

Usando o operador **IN**:

```
SELECT * FROM Produto
WHERE Tipo_Prod IN ('Sala', 'Quarto')
```

Considerando o nosso exemplo a seguir:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

Qualquer um dos dois comandos anteriores, quando executados nesse nosso exemplo, produziria este resultado:

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00

3.3 – A cláusula WHERE com o operador NOT IN

Ao utilizar o operador **NOT IN**, como mostra o comando seguinte, observemos os dados que são obtidos como resposta:

```
SELECT * FROM Produto
WHERE Tipo_Prod NOT IN ('Sala', 'Quarto')
```

Os mesmos dados poderiam ser obtidos com o operador **AND**, utilizando o sinal de comparação o (diferente).

```
SELECT * FROM Produto
WHERE Tipo_Prod <> 'Sala'
AND Tipo_Prod <> 'Quarto'
```

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
5	Fogão	Cozinha	400	700.00
10	Cadeira	Cozinha	1000	150.00

3.4 – A cláusula WHERE com o operador BETWEEN

O operador **BETWEEN** é utilizado para selecionar uma faixa de valores a serem atingidos por um determinado comando.

Para exemplificar, suponhamos que o usuário deseje selecionar todos os produtos cuja quantidade em estoque esteja entre 200 e 400 unidades. Para tanto, poderíamos escrever o comando a seguir:

```
SELECT * FROM Produto
WHERE Qtd_Prod BETWEEN 200 AND 400
```

Poderíamos também obter os mesmos dados escrevendo o comando **SELECT** da seguinte forma:

```
SELECT * FROM Produto
WHERE Qtd_Prod >= 200
AND Qtd_Prod <= 400
```

Considerando o conjunto de dados da tabela **Produto** apresentado a seguir, observemos quais serão os dados obtidos com a execução de um dos comandos anteriores,

Dados atuais da tabela **Produto**:

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

Resposta após a execução de um dos comandos **SELECT** anteriores:

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
9	Penteadeira	Quarto	300	300.00

3.5 – A cláusula WHERE com os operadores NOT BETWEEN

Se executarmos **SELECT** utilizando os operadores **NOT BETWEEN**, como mostra o exemplo a seguir, outros dados serão obtidos como resposta.

```
SELECT * FROM Produto
      WHERE Qtd_Prod NOT BETWEEN 200 AND 400
```

Poderíamos também obter os mesmos dados do comando anterior escrevendo o comando **SELECT** da seguinte forma:

```
SELECT * FROM Produto
      WHERE Qtd_Prod < 200 OR Qtd_Prod > 400;
```

Os dados resposta dos comandos anteriores são estes:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
10	Cadeira	Cozinha	1000	150.00

3.6 – A cláusula WHERE com o operador LIKE

O operador **LIKE** é utilizado para realizar pesquisas que utilizam como base colunas no formato caractere.

Ainda utilizando o exemplo da tabela **Produto**, novamente exibida a seguir, suponhamos que o usuário precisasse obter os dados de todas as colunas cujo nome de produto começasse com a letra P e terminasse com qualquer caractere. Para tanto, poderíamos utilizar o operador **LIKE**.

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

Observemos o comando:

```
SELECT * FROM Produto
WHERE Nome_Prod LIKE 'P%'
```

Observemos a resposta:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
7	Poltrona	Sala	600	332.75
9	Penteadeira	Quarto	300	300.00

O sinal de porcentagem (%) representa o caractere curinga. Ou seja, ele é utilizado para representar qualquer caractere e em qualquer quantidade.

Utilizando os dados da tabela **Produto**, observemos alguns exemplos de comandos que utilizam o operador **LIKE**:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

Selecione todos os produtos cujo tipo de produto comece com a letra **C**, e tenha a letra Z em qualquer posição:

```
SELECT * FROM PRODUTO
WHERE Tipo_Prod LIKE 'C%Z%'
```

Dados resposta:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
5	Fogão	Cozinha	400	700.00
10	Cadeira	Cozinha	1000	150.00

Selecione todos os produtos cujo nome comece com qualquer caractere, e tenha a letra M em qualquer posição.

```
SELECT * FROM Produto
WHERE Nome_Prod LIKE '%M%'
```

A resposta obtida será esta:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
2	Armário	Cozinha	700	300.00
4	Cama	Quarto	400	500.00

Selecione todos os produtos cujo nome e tipo contêm a sílaba **ar**

```
SELECT * FROM Produto
WHERE Nome_Prod LIKE '%ar%'
AND Tipo_Prod LIKE '%ar%'
```

A resposta obtida será esta:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
6	Guarda-Roupa	Quarto	500	1000.00

3.7 – A cláusula Where com os operadores NOT LIKE

Observemos quais são os dados obtidos como resposta ao utilizar os operadores **NOT LIKE** nos mesmos comandos executados anteriormente:

```
SELECT * FROM Produto
WHERE Nome_Prod NOT LIKE 'P%'
```

Resposta obtida:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
2	Armário	Cozinha	700	300.00
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00

```
SELECT * FROM Produto
WHERE Tipo.Prod NOT LIKE 'C%Z%'
```

Resposta obtida:

Produto

Cód_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
3	Sofá	Sala	300	1452.00
4	Cama	Quarto	400	500.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00

```
SELECT * FROM Produto
WHERE Nome_Prod NOT LIKE '%M%'
```

Resposta obtida:

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
3	Sofá	Sala	300	1452.00
5	Fogão	Cozinha	400	700.00
6	Guarda-Roupa	Quarto	500	1000.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
9	Penteadeira	Quarto	300	300.00
10	Cadeira	Cozinha	1000	150.00


```
SELECT * FROM Produto
WHERE Nome_Prod NOT LIKE '%ar%'
AND Tipo_Prod NOT LIKE '%ar%'
```

Resposta obtida:

Produto

Cod_Prod	Nome_Prod	Tipo_Prod	Qtd_Prod	Val_Prod
1	Geladeira	Cozinha	500	1650.00
3	Sofá	Sala	300	1452.00
5	Fogão	Cozinha	400	700.00
7	Poltrona	Sala	600	332.75
8	Estante	Sala	800	532.40
10	Cadeira	Cozinha	1000	150.00

Obs: O caracter '_' (underline) é um caracter curinga de uma posição.

Para fixação dos comandos acima, faça os exercícios abaixo:

Criar a tabela **Brinquedo** no database **Brinquedos** e inserir os dados a seguir:

Brinquedo

COD_BRINQ INT	NOM_BRINQ CHAR(50)	IDA_BRINQ TINYINT	QTD_BRINQ INT	VAL_BRINQ DECIMAL(10,2)
1	Amiguinha	7	100	150.00
2	Bebê Balancinho	3	200	30.00
3	Bebê Chuveirinho	3	50	25.00
4	Bolinha de Sabão	5	40	30.00
5	Emília	7	70	60.00
6	Hora do Papá	1	5	50.00
7	Frutinha Chaveirinho	2	30	20.00
8	Banco Imobiliário	10	10	100.00
9	Cai não Cai	3	15	35.00
10	Cara a Cara	4	16	28.00
11	Cilada	10	17	25.00
12	Combate	7	19	40.00
13	Detetive	7	20	50.00
14	A Galinha Magricela	7	21	90.00
15	Puxa-Puxa Batatinha	2	22	150.00
16	Robô Baby	2	23	200.00
17	Action Man	3	26	150.00
18	Batcicle	5	30	180.00
19	Batman	3	40	50.00
20	Super-man	4	50	50.00
21	Maga Patalógica	4	66	100.00
22	Paty Gatinha	3	80	20.00
23	Goleiro Maluco	5	92	20.00
24	Bichinhos do Bebê	6	100	20.00
25	Caixa-Encaixa	7	200	35.00
26	Guitarrinha do Bebê	4	200	80.00
27	Sr. e Sra.Cabeça de Batata	6	300	150,00
28	Ponteirinho	4	100	50.00
29	Bate-Car	5	200	30.00
30	Capot-Car	5	200	30.00

2 - Aumentar em 10 reais o valor de todos os brinquedos;

3 - Aumentar em 10% o valor de todos os brinquedos cujo nome comece com a letra **C**;

4 - Aumentar em 10 unidades a quantidade de todos os brinquedos cujo valor seja inferior a 50.00;

5 - Diminuir em um ano o valor da coluna **Ida_Brinq** de todos os brinquedos cujo nome contenha a palavra **Bebê**;

6 - Alterar a idade para um ano a menos e o valor para 20 reais a mais de todos os brinquedos cujo nome contenha a palavra **Gatinha**;

7 - Aumentar em 5 reais o valor de todos os brinquedos cujo nome contenha a palavra **Batata** ou **Batatinha**;

8 - Aumentar em 20 reais o valor dos brinquedos, diminuir em 5 unidades a quantidade dos brinquedos e aumentar 1 ano na idade de todos os brinquedos que estão com o valor Inferior ou igual a 30.00, que estão com a quantidade em estoque superior ou igual a 100, cujo nome comece com a palavra B.

9 - Aplicar um desconto de 10% (*0.9) no valor de todos os produtos cujo valor esteja entre 100.00 e 150.00;

10 - Excluir o brinquedo de código 10;

11 - Excluir os brinquedos cuja idade seja maior ou igual a 10 anos;

12 - Exibir todos os brinquedos cujo nome tenha até 6 letras;

13 - Exibir os brinquedos cujo nome tenha a palavra **Papá** ou **Galinha**;

14 - Exibir todos os brinquedos de idade inferior ou igual a três anos;

15 - Descrever o que faz cada comando ou cláusula a seguir:

SELECT =

INSERT =

UPDATE =

DELETE =

4 – Cláusula ORDER BY

Introdução

A cláusula **ORDER BY** é utilizada quando queremos ordenar os dados de uma tabela ao exibi-los.

Para demonstrar a utilização da cláusula **ORDER BY**, será utilizada a tabela **Material** criada no database **Materiais** , exibida a seguir:

Material

Cod_Mat	Tp_Mat	Nome_Mat	Qtd_Estq	Val_Unit
1	A	Caderno Espiral	100	12.50
2	A	Lápis Preto	500	1.50
3	A	Lápis de Cor	600	7.50
4	A	Caneta Azu!	800	5.50
5	B	Caneta Preta	700	5.50
6	B	Caderno Brochura	100	10.00
7	B	Borracha Mada	900	5.50
8	B	Borracha Colorida	1000	1.50
9	C	Apontador Colorido	2000	5-50
10	C	Régua de Madeira	3500	9.00
11	C	Régua de Plástico	2586	10.00
12	C	Folhas de Papel	2587	5.50

```
USE MATERIAIS
GO
CREATE TABLE Material
(
    COD_MAT INT,
    TP_MAT CHAR(1) ,
    NOME_MAT CHAR(50) ,
    QTD_ESTQ INT,
    VAL_UNIT DECIMAL(10,2)
)
GO
INSERT Material
VALUES(1, 'A', 'Caderno Espiral', 100,12.50)
```

Inserir os demais itens.

A cláusula **ORDER BY** é utilizada no comando **SELECT** para que se realize a ordenação dos dados a serem exibidos.

Podemos exibir os dados escolhendo o nome da coluna a ser ordenada ou o número referente à posição da coluna no **SELECT**.

Ordenando por uma coluna

- Escolhendo a ordenação pelo nome da coluna

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Nome_Mat
```

- Escolhendo a ordenação pela posição da coluna no SELECT

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 3
```

Ordenando por mais de uma coluna

Se for preciso, poderemos também ordenar os dados através de duas ou mais colunas.

Observe o exemplo seguinte, que ordena os dados da tabela **Material** por tipo de material e por valor:

- Escolhendo a ordenação pelo nome da coluna

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Tipo_Mat, Val_Unit
```

- Escolhendo a ordenação pela posição da coluna no SELECT

```
SELECT Tipo_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 1, 2
```

4.1 - ORDER BY ASC/DESC

Podemos também utilizar as opções **ASC** e **DESC** na cláusula **ORDER BY**.

A opção **ASC** já é o default, portanto, quando não for especificada **ASC** ou **DESC**, por default estará especificada a opção **ASC**. Essa opção é utilizada para que a cláusula **ORDER BY** ordene os dados de forma crescente, ou seja, ascendente.

DESC é a opção utilizada para que a cláusula **ORDER BY** ordene os dados de forma decrescente, ou seja, descendente.

Sendo assim, nas queries escritas anteriormente, para obtermos os mesmos dados, na mesma ordenação já apresentada, poderíamos acrescentar a opção **ASC** (que já é o default):

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Nome_Mat ASC
```

Ou

```
SELECT Tipo_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 3 ASC
```

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Tp_Mat ASC, Val_Unit ASC
```

Ou

```
SELECT Tipo_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 1 ASC, 2 ASC
```

Utilizando a opção **DESC**, observemos a resposta da execução dos comandos:

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Nome_Mat DESC
```

Ou

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 3 DESC
```

Observemos agora a ordenação dos dados por duas colunas de forma decrescente:

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Tp_Mat DESC, Val_Unit DESC
```

Ou

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 1 DESC, 2 DESC
```

Podemos também escrever comandos que ordenem os dados por meio de mais de uma coluna, sendo que uma delas apresenta uma ordenação crescente e outra decrescente. Observemos:

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY Tp_Mat ASC, Val_Unit DESC
```

Ou

```
SELECT Tp_Mat, Val_Unit, Nome_Mat
FROM Material
ORDER BY 1 ASC, 2 DESC
```

5 – A cláusula TOP

Essa cláusula permite que o usuário escolha o número de linhas a partir da primeira que deseja selecionar.

Se precisássemos, por exemplo, obter da tabela Material as 5 primeiras linhas, deveríamos escrever este comando:

```
SELECT TOP 5 * FROM Material
```

5.1 – A cláusula TOP com ORDER BY

A cláusula **TOP** pode ser muito útil com a cláusula **ORDER BY**. No exemplo da tabela Material, poderíamos querer obter os 3 primeiros produtos mais baratos ou, por exemplo, os 2 primeiros produtos mais caros. Para tanto, poderíamos escrever os dois comandos a seguir:

- Os três primeiros produtos mais baratos:

```
SELECT TOP 3 * FROM Material
ORDER BY Val_Unit ASC
```

- Os dois primeiros produtos mais caros:

```
SELECT TOP 2 * FROM Material
ORDER BY Val_Unit DESC
```

Material

Cód_Mat	Tp_Mat	Nome_Mat	Qtd_Estq	Val_Unit
2	A	Lápis Preto	500	1.50
8	B	Borracha Colorida	1000	1.50
7	B	Borracha Macia	900	5.50
5	B	Caneta Preta	150	5.50
4	B	Caneta Azul	800	5.50
9	C	Apontador Colorido	2000	5.50
12	C	Folha de Papel	2587	5.50
3	A	Lápis de Cor	600	7.50
10	C	Régua de Madeira	3500	9.00
11	C	Régua de Plástico	2586	10.00
6	B	Caderno Brochura	200	10.00
1	A	Caderno Espiral	100	12.50

O comando a seguir exibe os três primeiros produtos da tabela **Material**, ordenados em ordem crescente.

```
SELECT TOP 3 * FROM Material
ORDER BY Val_Unit ASC
```

Cód_Mat	Tipo_Mat	Nome_Mat	Qtd_Estq	Val_Unit
2	A	Lápis Preto	500	1.50
8	B	Borracha Colorida	1000	1.50
7	B	Borracha Macia	900	5.50

6 – Integridade e Consistência dos Dados

6.1 – Introdução

Um sistema de banco de dados deve sempre poder fornecer informações confiáveis a seus usuários. Para tanto, o administrador de dados deve filtrar certas ações sobre estes dados para evitar que possíveis erros possam ser cometidos sobre os mesmos.

Um database deve poder crescer de maneira independente em relação aos dados. Ou seja, seu sistema deve ser capaz de receber todas as informações necessárias sem ter que sofrer reformas constantes.

6.2 – Regras de integridade e consistência

Para exemplificar essas situações, usaremos o exemplo seguinte:

Suponhamos que uma determinada empresa precisasse armazenar os dados a seguir sobre seus funcionários:

Nome_Func	Sexo_Func	Sal_Func
Antônio Antunes	M	800.00
Pedro Paulo Pereira	M	900.00
Antônio Antunes	M	800.00

6.3 – Chaves primárias

É possível existir duas pessoas com o mesmo nome, mesmo sexo e mesmo salário trabalhando na mesma empresa? A resposta para essa pergunta é SIM. Claro que é possível existir na mesma empresa duas pessoas com nome, sexo e salários iguais.

Então, o que deveria ser feito para aplicarmos um aumento salarial a apenas um dos Antônio Antunes relacionados?

A pergunta anterior ficaria melhor formulada da seguinte maneira: como podemos identificar cada funcionário como único para aplicar-lhes um aumento salarial diferenciado?

Nesse caso, poderíamos solucionar o problema acrescentando na tabela uma coluna que recebesse um valor diferente para cada pessoa. Observemos:

Cod_Func	Nome_Func	Sexo_Func	Sal_Func
1	Antônio Antunes	M	800.00
2	Pedro Paulo Pereira	M	900.00
3	Antônio Antunes	M	800.00

Ao acrescentar a coluna **Cod_Func** (Código do Funcionário) na tabela anterior, torna-se possível identificar cada funcionário como único e aplicar-lhe o respectivo aumento salarial. Mas, observemos a tabela seguinte:

Cod_Func	Nome_Func	Sexo_Func	Sal_Func
1	Antônio Antunes	M	800.00
2	Pedro Paulo Pereira	M	900.00
1	Antônio Antunes	M	800.00

Imaginemos que, por uma grande coincidência, o digitador tenha errado e registrado o código 1 para os dois Antônio Antunes. Assim, voltamos à estaca zero. Novamente, não temos como identificar cada linha da tabela como única, pois os usuários podem digitar dados já existentes na tabela.

Então, para garantir que seja possível identificar cada linha como única na tabela **Funcionário**, além de acrescentar a coluna **Cod_Func**, devemos garantir que essa coluna não receba um valor já existente na tabela. Mesmo que o usuário informe um valor repetido, o sistema deve rejeitá-lo. Para garantir que uma coluna não aceite valores repetidos, devemos colocar sobre esta a chave primária.

Chave primária é uma regra de integridade e consistência dos dados que:

- não aceita valores repetidos na coluna onde ela existir;
- deve fazer parte da integridade referencial estabelecida entre duas tabelas que se relacionam (assunto tratado mais adiante).

Assim, colocando a chave primária na coluna **Cod_Func**, poderemos garantir que, na tabela **Funcionário**, cada linha será identificada como única.

Funcionario

Cod_Func Chave Primária	Nome_Func	Sexo_Func	Sal_Func
1	Antônio Antunes	M	800.00
2	Pedro Paulo Pereira	M	900.00
3	Antônio Antunes	M	800.00

6.4 – Chaves primárias secundárias ou chaves únicas

Os usuários cometem erros? A resposta para essa pergunta é SIM.

Portanto, para complicar um pouco mais, suponhamos que o **Usuário A** não sabia que o **Usuário B** já havia registrado o funcionário Pedro Paulo Pereira na tabela e o registrou novamente, quando na realidade os dois funcionários Pedro Paulo Pereira são a mesma pessoa. Como impedir esse tipo de engano?

Não seria justo (e nem prático) fazer com que todos os usuários que fossem cadastrar dados em uma tabela tivessem sempre que, antes de registrar uma informação, ler a tabela toda para garantir que não estão registrando novamente uma informação já existente.

Observemos a tabela a seguir:

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	M	900.00
3	Antônio Antunes	98.589.696	M	800.00
4	Pedro Paulo Pereira	47.258.569	M	900.00

Existem duas pessoas com o mesmo **RG** (Registro Geral)? Não. Cada pessoa deve ter o seu número de **RG** particular. Logo, se o usuário quisesse registrar o mesmo **RG** para mais de um funcionário, seria fácil perceber seu erro, uma vez colocada nesta coluna a chave primária.

Esse raciocínio está certo, exceto pelo fato de que em uma tabela é possível existir apenas uma chave primária, e esta já fora colocada na coluna **Cod_Func**. A coluna chave primária poderá ser composta de mais de uma coluna (assunto tratado mais adiante), mas só poderá haver uma chave primária na tabela. Para resolver essa situação, existem as chamadas chaves primárias secundárias, também conhecidas como chaves únicas.

Desse modo, tudo fica mais fácil, bastando que coloquemos a chave primária na coluna **Cod_Func**, que será utilizada para identificar cada linha como única, e a chave primária secundária (chave única), na coluna **RG_Func**, impedindo que valores já existentes na tabela sejam registrados novamente.

6.5 – Regras de validação

Observemos agora os valores das colunas **Sexo_Func** e **Sal_Func** referentes ao funcionário de código 2:

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	X	900.00
3	Antônio Antunes	98.589.696	M	800.00

O usuário poderia digitar valores errados como, por exemplo, na coluna Sexo registrar a letra X e na coluna salário registrar um valor negativo? A resposta para essa pergunta também é SIM.

Nesse caso, qual seria a resposta obtida quando tivéssemos que executar uma query como esta:

```
SELECT * FROM Funcionario
WHERE Sexo Func = 'M'
```

A resposta seria:

Cod_Func	Nome_Func	RG_Func	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	M	800.00
3	Antônio Antunes	98.589.696	M	800.00

Apenas dois funcionários seriam apresentados como resposta, mas isso não representa a realidade, pois na tabela existem três funcionários do sexo masculino. Nesse caso podemos dizer que os dados do sistema não são confiáveis, pois nem todas as respostas obtidas representam seu real conteúdo.

Poderíamos dizer o mesmo sobre o salário dos funcionários. Se tivéssemos que calcular o valor total gasto em salários com os funcionários, também não obteríamos uma resposta correta. Vejamos:

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	X	-900.00
3	Antônio Antunes	98.589.696	M	800.00
Soma Total dos Salários				700.00

Sendo assim, para garantirmos que os dados são confiáveis, devemos colocar certas regras de validação nestas colunas, impedindo os possíveis enganos dos usuários. Observemos:

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func Só aceitar (F ou M)	Sal_Func Só aceitar valores positivos
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	M	900.00
3	Antônio Antunes	98.589.696	M	800.00

Agora nossos dados são íntegros e consistentes, porque só aceitam valores que representam a realidade, sem repetições indesejáveis. Neste ponto, vamos supor que precisássemos também armazenar dados sobre os dependentes de cada funcionário

6.6 – Chaves estrangeiras

Talvez pudéssemos pensar em acrescentar mais uma coluna na tabela anterior. Vejamos:

Funcionario

Cod_Func	Nome_Func	RG_Func	Nome_Dep	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	Maria Antunes	M	800.00
2	Pedro Paulo Pereira	47.258.569	Maria Pereira	M	900.00
3	Antônio Antunes	98.589.696	José Antunes	M	800.00

Se cada funcionário pudesse ter apenas um dependente e fosse necessário armazenar apenas o nome deste no sistema, então essa tabela estaria correta. Mas, sabemos que na realidade cada funcionário poderá ter vários dependentes. Sabemos também que é necessário armazenar outras informações a respeito de cada um dos dependentes que os funcionários possuem, como data de nascimento, sexo, etc. Para solucionarmos essa questão, devemos separar os dados dos dependentes em outra tabela. Observemos:

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	M	900.00
3	Antônio Antunes	98.589.696	M	800.00

Dependente

Cod_Dep Chave Primária	Nome_Dep	DataNasc_Dep	Sexo_Dep
1	Maria Antunes	01/02/88	F
2	Maria Pereira	05/05/99	F
3	José Antunes	06/07/98	M
4	Josefa Antunes	06/07/98	F

Muito bem, agora é possível registrarmos no sistema uma quantidade qualquer de dependentes para cada funcionário. A coluna **Cod_Dep** não aceita valores repetidos, e a coluna **Sexo_Dep** só aceita as letras M ou F, que representam masculino e feminino, respectivamente. Conseguiríamos, no entanto, selecionar todos os dependentes que o funcionário de código 3 possui? A resposta para essa pergunta é NÃO. Então, como fazer para identificarmos todos os dependentes de cada funcionário?

Para resolvermos essa questão basta acrescentarmos na tabela **Dependente** uma coluna da tabela **Funcionário**. Observemos:

Dependente

Cod_Dep Chave Primária	Cód_Func	Nome_Dep	DataNasc_Dep	Sexo_Dep
1	1	Maria Antunes	01/02/88	F
2	2	Maria Pereira	05/05/99	F
3	3	José Antunes	06/07/98	M
4	3	Josefa Antunes	06/07/98	F

Agora fica fácil sabermos quais são os dependentes de cada funcionário. Vejamos:

```
SELECT * FROM Dependente
WHERE Cod Func = 1
```

Resposta da query anterior:

Cod_Dep	Cód_Func	Nome_Dep	DataNasc_Dep	Sexo_Dep
1	1	Maria Antunes	01/02/88	F

```
SELECT * FROM Dependente
WHERE Cod Func = 2
```

Resposta da query anterior:

Cod_Dep	Cód_Func	Nome_Dep	DataNasc_Dep	Sexo_Dep
2	2	Maria Pereira	05/05/99	F

```
SELECT * FROM Dependente
WHERE Cod Func = 3
```

Resposta da query anterior:

Cod_Dep	Cód_Func	Nome_Dep	DataNasc_Dep	Sexo_Dep
3	3	José Antunes	06/07/98	M
4	3	Josefa Antunes	06/07/98	F

Ainda temos aqui um problema. Como já sabemos, os usuários costumam cometer certos enganos não-propositais. Falta ainda prevermos um outro tipo de engano possível de acontecer. Vejamos o exemplo:

Cod_Dep Chave Primária	Cód_Func	Nome_Dep	DataNasc_Dep	Sexo_Dep
1	1	Maria Antunes	01/02/88	F
2	2	Maria Pereira	05/05/99	F
3	3	José Antunes	06/07/98	M
4	3	Josefa Antunes	06/07/98	F
5	1000	Sônia Gost	04/01/00	F

Qual é o nome do funcionário responsável pelo dependente 5?

O que acontece aqui é que nosso sistema não está mantendo a integridade referencial entre as tabelas **Funcionário e Dependente**. Por esse mesmo motivo, se algum usuário tentar excluir da tabela **Funcionário** algum funcionário que tenha dependentes, conseguirá, e assim teremos dependentes órfãos em nosso sistema.

Para evitarmos esse tipo de inconsistência dos dados, deveremos colocar na coluna **Cod_Func** da tabela **Dependente** a regra chamada chave estrangeira.

Observemos agora as duas tabelas com todas as regras necessárias para que se mantenha a integridade e a consistência dos dados do sistema:

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func	Sal_Func
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	M	900.00
3	Antônio Antunes	98.589.696	M	800.00

Dependente

Cod_Dep Chave Primária	Cód_Func Chave Estrangeira	Nome_Dep	DataNasc_Dep	Sexo_Dep
1	1	Maria Antunes	01/02/88	F
2	2	Maria Pereira	05/05/99	F
3	3	José Antunes	06/07/98	M
4	3	Josefa Antunes	06/07/98	F

6.7 – Valor padrão (Default)

Em certas colunas dessas duas tabelas poderia ser colocada outra regra, o valor padrão. Esse valor padrão é mais uma facilidade para o usuário, pois, se este deixar de informar um valor para essas colunas, o sistema assume o valor da regra como padrão. Nesse caso, temos na tabela **Funcionario**:

- O valor **F** como padrão da coluna **Sexo_Func**;
- O valor **415.00** como padrão da coluna **Sal_Func**.

Na tabela **Dependente**:

- O valor **M** como padrão da coluna **Sexo_Dep**

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func Padrão = F	Sal_Func Padrão = 415.00
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	M	900.00
3	Antônio Antunes	98.589.696	M	800.00

Dependente

Cod_Dep Chave Primária	Cód_Func Chave Estrangeira	Nome_Dep	DataNasc_Dep	Sexo_Dep Padrão = M
1	1	Maria Antunes	01/02/88	F
2	2	Maria Pereira	05/05/99	F
3	3	José Antunes	06/07/98	M
4	3	Josefa Antunes	06/07/98	F

6.8 – Constraints

A palavra constraint significa restrição. Uma constraint é um objeto utilizado para impor regras de integridade e consistência nas colunas das tabelas de um sistema.

Tipo de Integridade	Tipo de Constraint
Chave Primária	Constraint Primary Key
Chave Primária Secundária ou chave Única	Constraint Unique
Chave Estrangeira	Constraint Foreign Key e Constraint References
Regras de Validação	Constraint Check
Valor Padrão	Constraint Default

6.9 – Valores nulos e não-nulos

Em nosso sistema poderia existir um funcionário que não tivesse um código, não tivesse um nome, não tivesse um RG, não tivesse sexo e/ou não tivesse um salário?

Funcionario

Cod_Func Chave Primária	Nome_Func	RG_Func Chave Única	Sexo_Func Padrão = F	Sal_Func Padrão = 350.00
1	Antônio Antunes	15.253.454	M	800.00
2	Pedro Paulo Pereira	47.258.569	M	900.00
3	Antônio Antunes	98.589.696	M	800.00
NULL	NULL	NULL	NULL	NULL

Digamos que se deseje que apenas a coluna **Sal_Func** aceite valores nulos e que se permita que funcionários sejam inseridos sem salário nenhum, pois esse valor deverá ser atualizado em algum outro momento no sistema. Mas todos os funcionários da sua tabela devem ter um código, um nome, um RG e um sexo.

Quando uma coluna aceita valores nulos, significa que, dependendo do datatype que ela possua, essa coluna ocupará espaço no seu database, mas não possuirá nenhum valor.

Valor nulo significa ausência de valor. Sendo assim, como regra de integridade e consistência dos dados, deveremos também definir a **nullability** ("nulabilidade") das colunas de nossa tabela. Ou seja, deveremos definir se cada coluna aceitará valor nulo ou não.

No MSSQL, quando criamos uma tabela e não definimos explicitamente a nulabilidade de cada coluna, estas aceitarão valores nulos, exceto aquelas que forem definidas como chave primária.

6.10 – Datatypes

Os datatypes, já citados anteriormente, também servem para manter a integridade e a consistência dos dados, pois limitam os valores a serem inseridos em uma determinada coluna. Por exemplo, não é possível inserir um caractere em uma coluna numérica, assim como não é possível inserir um valor qualquer em uma coluna no formato data.

Vejamos como ficam as regras de integridade e consistência das colunas das duas tabelas:

Funcionario

Coluna	Regra
Cód_Func	Int Chave Primária Não-nula
Nome_Func	Char(50) Não-nula
RG_Func	Char(15) Chave Única Não-nula
Sexo_Func	Char(1) Valor Padrão = F
Sal_Func	Decimal(10,2) Valor Padrão = 350.00 Aceita valor nulo

Dependente

Coluna	Regra
Cod_Dep	int Chave Primária Não-Nula
Cod_Func	Int Chave Estrangeira Não-Nula
Nome_Dep	Char(50) Não-Nula
DataNasc_Dep	Datetime Não-Nula
Sexo_Dep	Char(1) Valor Padrão = M Não-Nula

6.11 – Criando as tabelas com todas as regras de integridade e consistência**A tabela Funcionario:**

```
USE TURMA3X  
GO
```

```

CREATE TABLE FUNCIONARIO
(
    COD_FUNC      INT          NOT NULL,
    NOME_FUNC     CHAR(50)     NOT NULL,
    RG_FUNC       CHAR(15)     NOT NULL,
    SEXO_FUNC     CHAR(1)      NOT NULL      DEFAULT 'F',
    SAL_FUNC      DECIMAL(10,2) NOT NULL      DEFAULT 415.00,
    CONSTRAINT PK_FUNC PRIMARY KEY (COD_FUNC),
    CONSTRAINT UQ_FUNC UNIQUE (RG_FUNC)
)

```

Observemos que cada constraint recebeu um nome – que são opcionais no MSSQL.

As palavras **PK_Func**, **UQ_Func** são nomes de constraints. Esses nomes não têm que seguir este padrão. Podemos atribuir o nome que quisermos para qualquer constraint que criarmos, lembrando que cada nome deverá ser único no banco de dados. Por convenção, utiliza-se as iniciais para:

PK	PRIMARY KEY
UQ	UNIQUE
FK	FOREIGN KEY
CK	CHECK

A tabela Dependente:

```

USE TURMA3X
GO
CREATE TABLE DEPENDENTE
(
    COD_DEP      INT          NOT NULL,
    COD_FUNC     INT          NOT NULL,
    NOME_DEP     CHAR(50)     NOT NULL,
    DATANASC_DEP DATETIME     NOT NULL,
    SEXO_DEP     CHAR(1)      NOT NULL      DEFAULT 'M',
    CONSTRAINT PK_DEP PRIMARY KEY (COD_DEP),
    CONSTRAINT FK_DEP FOREIGN KEY (COD_FUNC)
        REFERENCES FUNCIONARIO (COD_FUNC)
)

```

6.12 – Regras das Constraints

Constraint Primary Key

- Toda tabela poderá ter apenas uma constraint de chave primária;
- A coluna a ser definida como chave primária não poderá ser definida como nula.

Constraint Foreign Key

- Para ser chave estrangeira em uma tabela B, esta coluna deverá antes ser chave primária na tabela A;
- Uma tabela poderá ter várias chaves estrangeiras;
- As colunas chave estrangeira poderão aceitar valores Nulos.

Constraint Unique

- Uma tabela poderá ter várias chaves únicas;
- As colunas chave única poderão aceitar valor único.

Constraint Check

- Uma tabela poderá ter várias **constraints check**;
- As colunas que contêm as **constraints check** poderão aceitar valores nulos dependendo da regra que especificarmos para elas.

Constraint Default

- Uma tabela poderá ter várias **constraints default**;
- As colunas que contêm **constraint Default** poderão aceitar valores nulos.

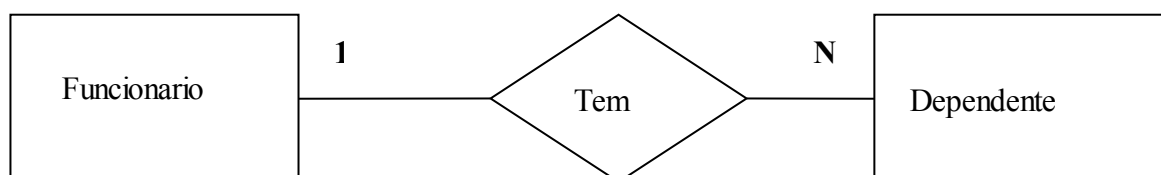
6.13 – Tipos de relacionamento

Para expor os tipos de relacionamento que o SQL Server pode gerenciar, será utilizado o Modelo Entidade Relacionamento - MER. Esse modelo permite-nos representar os relacionamentos existentes entre os dados de um sistema.

6.13.1 – Relacionamento 1:N

O relacionamento de 1:N é o que acontece entre as tabelas **Funcionário e Dependente**.

Note que um funcionário pode ter muitos dependentes, mas um dependente só poderá pertencer a um funcionário:



Funcionario

Cod_Func	Nome_Func	Sexo_Func	Sal_Func
1	Antônio Pereira de Albuquerque	M	2500.50
2	José Genuíno Geremias	M	600.00

Dependente

Cod_Dep	Cod_Func	Nome_Dep	Sexo_Dep
1	1	Antônio Pereira de Albuquerque Filho	M
2	1	Maria Cristina Pereira de Albuquerque	F
3	1	Marcio Pereira de Albuquerque	M

Observemos que o funcionário de código 1 tem 3 dependentes, e o funcionário de código 2 não tem dependentes. Notemos também que cada dependente pertence a apenas um funcionário, nesse caso, ao funcionário de código 1.

Os comandos para a criação das tabelas e para a inclusão de dados são os seguintes:

```
CREATE TABLE FUNCIONARIO
(
    COD_FUNC      INT          NOT NULL,
    NOME_FUNC     CHAR(50)     NOT NULL,
    SEXO_FUNC     CHAR(1)      NOT NULL,
    SALFUNC       DECIMAL(10,2) NOT NULL,
    CONSTRAINT PK_FUNC PRIMARY KEY (COD_FUNC)
)
GO
INSERT FUNCIONARIO VALUES
    (1, 'ANTÔNIO PEREIRA DE ALBUQUERQUE', 'M', 2500.50)
INSERT FUNCIONARIO VALUES
    (2, 'JOSÉ GENUÍNO GEREMIAS', 'M', 600.00)

CREATE TABLE DEPENDENTE
(
    COD_DEP  INT      NOT NULL,
    COD_FUNC INT      NOT NULL,
    NOME_DEP CHAR(50) NOT NULL,
    SEXO_DEP CHAR(1)  NOT NULL,
    CONSTRAINT PK_DEP PRIMARY KEY (COD_DEP),
    CONSTRAINT FK_FUNC FOREIGN KEY (COD_FUNC)
        REFERENCES FUNCIONARIO (COD_FUNC)
)

INSERT DEPENDENTE VALUES
    (1, 1, 'ANTÔNIO PEREIRA DE ALBUQUERQUE FILHO', 'M')
INSERT DEPENDENTE VALUES
    (2, 1, 'MARIA CRISTINA PEREIRA DE ALBUQUERQUE', 'F')
INSERT DEPENDENTE VALUES
```

(3,1,'MARCIO PEREIRA DE ALBUQUERQUE','M')

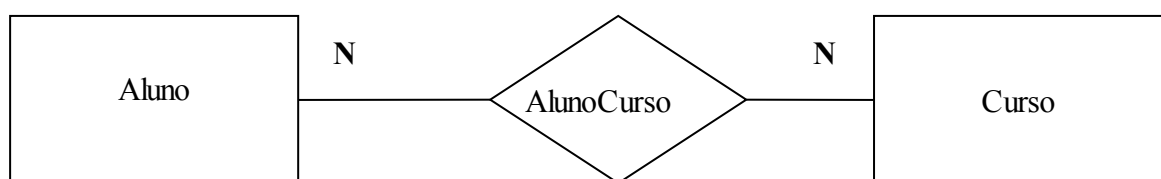
Lembrete: Regras de relacionamento de 1 :N

- As duas tabelas deverão ter colunas que contenham chaves primárias, mas é imprescindível que a tabela do lado 1 do relacionamento tenha uma coluna com a **constraint primary key**;
Nota: A chave primária não aceita valores repetidos.
- A tabela do lado N deverá ter uma coluna referente a tabela do lado 1 para estabelecer o relacionamento entre as duas, e esta coluna deverá receber a constraint de chave estrangeira.
Nota: A chave estrangeira aceita valores repetidos.

6.3.2 – Relacionamento N:N

O relacionamento de **N:N** é o que acontece entre as tabelas **Aluno e Curso**.

Notemos que 1 aluno pode fazer muitos cursos e um curso pode ser feito por muitos alunos



Aluno

Cod_Aluno	Nome_Aluno	Sexo_Aluno
1	Ronaldo Pompilho	M
2	Amaro Amarildo	M
3	Márcia Garcia	F

Curso

Cod_Curso	Nome_Curso
1	Matemática
2	Física
3	Química
4	Geografia

AlunoCurso

Cod_Aluno	Cod_Curso
1	1
1	2
1	3
1	4
2	1
2	4
3	1
3	3

```

CREATE TABLE Aluno
(
    Cod_Aluno int      Not Null,
    Nome_Aluno char(50) Not Null,
    Sexo_Aluno char(1) Not Null,
    Constraint PK_Aluno Primary Key (Cod_Aluno)
)
GO
INSERT Aluno VALUES
    (1, 'Ronaldo Pompilho', 'M')
INSERT Aluno VALUES
    (2, 'Amaro Amarildo', 'M')
INSERT Aluno VALUES
    (3, 'Márcia Garcia', 'F')

CREATE TABLE Curso
(
    Cod_Curso int      Not Null,
    Nome_Curso char(30) Not Null,
    Constraint PK_Curso Primary Key (Cod_Curso),
    Cosntraint UQ_Curso Unique (Nome_Curso)
)
GO
INSERT Curso VALUES(1, 'Matemática');
INSERT Curso VALUES(2, 'Física');
INSERT Curso VALUES(3, 'Química');
INSERT Curso VALUES(4, 'Geografia');

CREATE TABLE AlunoCurso
(
    Cod_Aluno int      Not Null,
    Cod_Curso int      Not Null,
    Constraint PK_AC Primary Key (Cod_Aluno, Cod_Curso),
    Constraint FK_AC1 Foreign Key (Cod_Aluno)
        References Aluno (Cod_Aluno),
    Constraint FK_AC2 Foreign Key (Cod_Curso)

```

```

References Curso (Cod_Curso)
)
GO
INSERT AlunoCurso VALUES (1, 1)
INSERT AlunoCurso VALUES (1, 2)
INSERT AlunoCurso VALUES (1, 3)
INSERT AlunoCurso VALUES (1, 4)
INSERT AlunoCurso VALUES (2, 1)
INSERT AlunoCurso VALUES (2, 4)
INSERT AlunoCurso VALUES (3, 1)
INSERT AlunoCurso VALUES (3, 3)

```

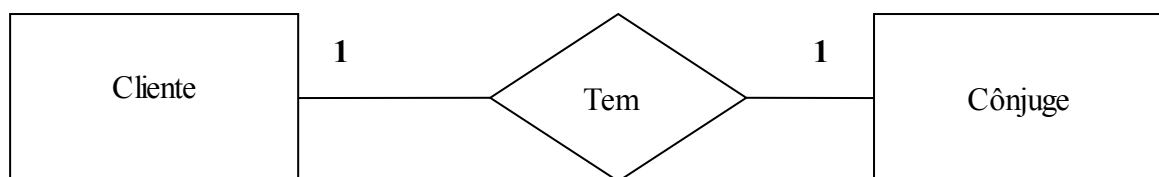
Lembrete: Regras de relacionamento de N:N

- Em um relacionamento de **N:N** deve ser criada uma terceira tabela para relacionar as outras duas;
- As duas tabelas deverão ter colunas que sejam chaves primárias;
- A terceira tabela deverá ter como chaves estrangeiras as colunas chave primária das outras duas tabelas;
- Na terceira tabela as duas chaves estrangeiras poderão formar uma chave primária composta.

6.3.3 – Relacionamento 1:1

O relacionamento de 1:1 é o que acontece entre as tabelas **Cliente e Cônjuge**.

Notemos que 1 cliente só poderá ter 1 cônjuge e 1 cônjuge só poderá pertencer a 1 cliente.



Cliente

Cod_Cli	Nome_Clie	Sexo_Cli
1	Sandra	F
2	Geraldo	M

Conjuge

Cod_Conj	Nome_Conj	Sexo_Conj
1	Marcos	M
2	Silvia	F

```
CREATE TABLE Cliente
(
    Cod_Cli    int        Not Null,
    Nome_Cli   char(50) Not Null,
    Sexo_Cli   char(1)  Not Null,
    Constraint PK_Cli Primary Key(Cod_Cli)
)
GO
INSERT Cliente VALUES
    (1, 'Sandra', 'F')
INSERT Cliente VALUES
    (2, 'Geraldo', 'M')

CREATE TABLE CONJUGE
(
    Cod_Cli    int        Not Null,
    Nome_Conj   char(50) Not Null,
    Sexo_Conj   char(1)  Not Null,
    Constraint PK_Conj Primary Key (Cod_Cli),
    Constraint FK_Conj Foreign Key(Cod_Cli)
        References Cliente(Cod_Cli)
)
GO
INSERT Conjuge VALUES (1, 'Marcos', 'M')
INSERT Conjuge VALUES (2, 'Silvia', 'F')
```

Para fixação dos comandos acima, faça os exercícios abaixo:

1 - No database **Turma3X**, criar a tabela e inserir os dados a seguir:

Candidato

Cod_Cand Int	Cod_Vaga Char(10)	Nome_Cand Char(20)	Nota_Cand Decimal(3,1)
1	AB_123	Zulmira	10.0
2	AB_345	Américo	6.0
3	AB_123	Tales	5.0
4	CD_001	Doutes	7.0
5	CD_001	Otto Otta	7.0
6	CD_002	Claudianela	7.0
7	EF_336	Xisto	8.0
8	EF_449	Antônio	8.0
9	EF_449	Ernesto	2.0
10	XPTO_01	Amélia	3.0
11	XPTO_02	Anatonildo	10.0
12	XPTO_03	Esmeralda	10.0
13	XPTO_04	Marcelo	9.0

2 - Exibir os dados da tabela **Candidato** ordenados de forma crescente, de acordo com a coluna **Nome Cand**.

3 - Exibir os dados da tabela **Candidato** ordenados de forma decrescente, de acordo com a coluna **Nome Cand**.

4 - Apresentar os dados da tabela **Candidato** ordenados de forma crescente, de acordo com a coluna **Nome_Cand**, mas exibir apenas os candidatos cujo código da vaga contenha a palavra **XPTO**.

5 - Apresentar os dados da tabela Candidatos ordenados de duas formas:

- Decrescente de acordo com a coluna **Nome Cand**;
- Crescente de acordo com a coluna **Cod_Vaga**.

6 - Apresentar os cinco primeiros candidatos inseridos na tabela.

7 - Apresentar os dados da tabela **Candidatos** ordenando-os de forma crescente de acordo com a nota obtida pelos candidatos.

8 - Apresentar o candidato que teve maior nota.

9 - Apresentar os dados dos 5 primeiros candidatos.

10 - Apresentar os dados do último colocado.

Parte B

- 1 - Neste exercício será criado um sistema simplificado de venda de cds, de acordo com o título de cada um. Para tanto, criar um database chamado DB_CDS com opções default;
2. Conectar-se no database **DB_CDS**.
3. Criar as tabelas especificadas a seguir com suas respectivas constraints:

Tabela 1

Nome da Tabela: Artista				
Nome Coluna	Descrição coluna	DataType	Nullability	Constraint
Cod_Art	Código Artista	Int	NOT NULL	Chave Primária
Nome_Art	Nome do artista ou nome Banda	VarChar(100)	NOT NULL	Chave única

Tabela 2

Nome da Tabela; Gravadora				
Nome_Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Grav	Código Gravadora	Int	NOT NULL	Chave Primária
Nome_Grav	Nome da Gravadora	Varchar(50)	NOT NULL	Chave Única

Tabela 3

Nome da Tabela: Categoria				
Nome_Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Cat	Código Categoria	Int	NOT NULL	Chave Primária
Nome_Cat	Nome Categoria	Varchar(50)	NOT NULL	Chave Única

Tabela 4

Nome da Tabela: Estado				
Nome_Coluna	Descrição coluna	DataType	Nullability	Constraint
Sigla_Est	Sigla Estado	Char(2)	NOT NULL	Chave Primária
Nome_Est	Nome Estado	Char(20)	NOT NULL	Chave Única

Tabela 5

Nome da Tabela: Cidade				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Cid	Código da Cidade	Int	NOT NULL	Chave Primária
Sigla_Est	Sigla do Estado	Char(2)	NOT Null	Chave Estrangeira referendando a tabela Estado
Nome_Cid	Nome da Cidade	Varchar(100)	NOT NULL	

Tabela 6

Nome da Tabela: Cliente				
Nome Coluna	Descrição Coluna	DataType	Nuliability	Constraint
Cod_Cli	Código do Cliente	Int	NOT NULL	Chave Primária
Cod_Cid	Código da Cidade	Int	NOT NULL	Chave Estrangeira referendando a tabela Cidade
Nome_Cli	Nome do Cliente	Varchar(100)	NOT NULL	
End_Cli	Endereço Cliente	Varchar(200)	NOT NULL	
Renda_Cli	Renda do Cliente	Decimal(10,2)	NOT NULL	default 0
Sexo_Cli	Sexo do Cliente	Char(1)	NOT NULL	Default 'F'

Tabela 7

Nome da Tabela: Conjuge				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Cli	Código do Cliente	Int	NOT NULL	Chave Primária e Chave estrangeira referenciando a tabela Cliente
Nome_Conj	Nome do Cônjuge	Varchar(100)	NOT NULL	
Renda_Conj	Renda do Cônjuge	Decimal(10,2)	NOT NULL	Default 0
Sexo_Conj	Sexo do Cônjuge	Char(1)	NOT NULL	Default 'M'

Tabela 8

Nome da Tabela: Funcionario				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Func	Código Funcionário	Int	NOT NULL	Chave Primária
Nome_Func	Nome Funcionário	Varchar(100)	NOT NULL	
End_Func	Endereço Func	Varchar(200)	NOT NULL	
Sal_Func	Salário Funcionário	Decimal(10,2)	NOT NULL	Default 0
Sexo_Func	Sexo Funcionário	Char(1)	NOT NULL	Default 'F'

Tabela 9

Nome da Tabela: Dependente				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Dep	Código Dependente	Int	NOT NULL	Chave Primária
Cod_Func	Código Funcionário	Int	NOT NULL	Chave Estrangeira referenciando a tabela Funcionário
Nome_Dep	Nome Dependente	Varchar(100)	NOT NULL	
Sexo_Dep	Sexo Dependente	Char(1)	NOT NULL	Default 'M'

Tabela 10

Nome da Tabela: Título				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Tit	Código do Título	Int	NOT NULL	Chave Primária
Cod_Cat	Código Categoria	Int	NOT NULL	Chave Estrangeira referenciando a tabela Categoria
Cod_Grav	Código Gravadora	Int	NOT NULL	Chave Estrangeira referenciando a tabela Gravadora
Nome_CD	Nome do CD	Varchar(100)	NOT NULL	Chave Única
Val_CD	Valor do CD	Decimal(10,2)	NOT NULL	
Otd_Estq	Quantidade de CD de cada Título em estoque	Int	NOT NULL	

Tabela 11

Nome da Tabela: Pedido				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Num_Ped	Número do Pedido	Int	NOT NULL	Chave Primária
Cod_Cli	Código do Cliente que está fazendo o Pedido	Int	NOT NULL	Chave Estrangeira referenciando a tabela Cliente
Cod_Func	Código do Funcionário que está atendendo o Pedido	Int	NOT NULL	Chave Estrangeira referenciando a tabela Funcionário
Data_Ped	Data de abertura do Pedido	Date	NOT NULL	
Val_Ped	Valor total do Pedido	Decimal(10,2)	NOT NULL	Default 0

Tabela 12

Nome da Tabela: Titulo_Pedido				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Num_Ped	Número do Pedido	Int	NOT NULL	Chave Estrangeira referenciando a tabela Pedido
Cod_Tit	Código do Título	Int	NOT NULL	Chave Estrangeira referenciando a tabela Título
Qtd_CD	Quantidade de CDs vendidos, de mesmo Título	Int	NOT NULL	Default 1
Val_CD	Valor do CD no momento da Venda	Decimal(10,2)	NOT NULL	

Tabela 13

Nome da Tabela: Titulo_Artista				
Obs: Esta tabela terá uma chave primária composta de duas colunas: (Cod_Tit e Cod_Art)				
Nome Coluna	Descrição Coluna	DataType	Nullability	Constraint
Cod_Tit	Código do Título	Int	NOT NULL	Chave Estrangeira referenciando a tabela Título
Cod_Art	Código do Artista	Int	NOT NULL	Chave Estrangeira referenciando a tabela Artista

4. Inserir os dados a seguir nas respectivas tabelas; (PEGAR A MASSA DE DADOS COM O PROFESSOR)

Artista

Cod_Art	Nome_Art
1	Marisa Monte
2	Gilberto Gil
3	Caetano Veloso
4	Milton Nascimento
5	Legião Urbana
6	The Beattes
7	Rita Lee

Gravadora

Cod_Grav	Nome_Grav
1	Polygram
2	EMI
3	Som Livre
4	Sony Music

Categoria

Nome_Cat	Nome_Cat
1	MPB
2	Trilha Sonora
3	Rock Internacional
4	Rock Nacional

Estado

Sigla_Est	Nome Estado
SP	São Paulo
MG	Minas Gerais
RJ	Rio de Janeiro
ES	Espírito Santo

Cidade

Cod_Cid	Nome_Cid	Sigla_Est
1	São Paulo	SP
2	Sorocaba	SP
3	Jundiaí	SP
4	Americana	SP
5	Araraquara	SP
6	Ouro Preto	MG
7	Cachoeiro Itapemirim	ES

Cliente

Cod_Cli	Cod_Cid	Nome_Cli	End_Cli	Renda_Cli	Sexo_Cli
1	1	José Nogueira	Rua A	1500.00	M
2	1	Angelo Pereira	Rua B	2000.00	M
3	1	Além Mar Paranhos	Rua C	1500.00	M
4	1	Catarina Souza	Rua D	892.00	F
5	1	Vagner Costa	Rua E	950.00	M
6	2	Antenor da Costa	Rua F	1582.00	M
7	2	Maria Amélia de Souza	Rua G	1152.02	F
8	2	Paulo Roberto Silva	Rua H	3250.00	M
9	3	Fátima Souza	Rua I	632.00	F
10	3	Joel da Rocha	Rua J	2000.00	M

Conjuge

Cod_Cli	Nome_Conj	Renda_Conj	Sexo_Conj
1	Carla Nogueira	2500.00	M
2	Emitia Pereira	5500.00	F
6	Altiva da Costa	3000.00	F
7	Carlos de Souza	3250.00	M

Funcionario

Cod_Func	Nome_Func	End_Func	Sal_Func	Sexo_Func
1	Vânia Gabriela Pereira	Rua A	2500.00	F
2	Norberto Pereira da Silva	Rua B	300.00	M
3	Olavo Linhares	Rua C	580.00	M
4	Paula da Silva	Rua D	3000.00	F
5	Rolando Rocha	Rua E	2000.00	M

Dependente

Cod_Dep	Cod_Func	Nome_Dep	Sexo_Dep
1	1	Ana Pereira	F
2	1	Roberto Pereira	M
3	1	Celso Pereira	M
4	3	Brisa Linhares	F
5	3	Mari Sol Linhares	F
6	4	Sônia da Silva	F

Título

Cod_Tit	Cod_Cat	Cod_Grav	Nome_CD	Val_CD	Qtd_Estq
1	1	1	Tribalistas	30.00	1500
2	1	2	Tropicália	50.00	500
3	1	1	Aquele Abraço	50.00	600
4	1	2	Refazenda	60.00	1000
5	1	3	Totalmente Demais	50.00	2000
6	1	3	Travessia	55.00	500
7	1	2	Courage	30.00	200
8	4	3	Legião Urbana	20.00	100
9	3	2	The Beatles	30.00	300
10	4	1	Rita Lee	30.00	500

Pedido

Num_Ped	Cod_Cli	Cod_Func	Data_Ped	Val_Ped
1	1	2	02/05/02	1500.00
2	3	4	02/05/02	50.00
3	4	5	02/06/02	100.00
4	1	4	02/02/03	200.00
5	7	5	02/03/03	300.00
6	4	4	02/03/03	100.00
7	5	5	02/03/03	50.00
8	8	2	02/03/03	50.00
9	2	2	02/03/03	2000.00
10	7	1	02/03/03	3000.00

Título_Artista

Cod_Tit	Cod_Art
2	2
3	2
4	2
5	3
6	4
7	4
8	5
9	6
10	7

Título_Pedido

Num_Ped	Cod_Tit	Qtd_CD	Val_CD
1	1	2	30.00
1	2	3	20.00
2	1	1	50.00
2	2	3	30.00
3	1	2	40.00
4	2	3	20.00
5	1	2	25.00
6	2	3	30.00
6	3	1	35.00
7	4	2	55.00

7 – Cláusula JOIN

Quando os dados estão armazenados em mais de uma tabela, muitas vezes, para transformá-los em informação, devemos associar essas tabelas.

As tabelas sempre serão associadas de duas em duas. Normalmente, a associação é feita entre a chave primária da tabela A e a chave estrangeira da tabela B. Poderemos associar mais de duas tabelas em um único comando, mas a associação será feita sempre de duas em duas tabelas.

Existem vários tipos de associação que podem ser escritos com a cláusula **JOIN** ou com a cláusula **WHERE**:

- Associação que obtém apenas os dados relacionados entre as duas tabelas;
- Associação que obtém os dados relacionados entre as duas tabelas e os não-relacionados que estiverem na tabela do lado esquerdo da cláusula **JOIN**;
- Associação que obtém os dados relacionados entre as duas tabelas e os não-relacionados que estiverem na tabela do lado direito da cláusula **JOIN**;
- Associação que obtém os dados relacionados entre as duas tabelas e também os não-relacionados;
- Associação que cruza os dados das duas tabelas. É o produto cartesiano.

7.1 – Obtendo apenas os dados relacionados – INNER JOIN

Cargo

Cod_Cargo Chave Primária	Nome_Cargo Chave Única
1	Diretor Presidente
2	Diretor de Departamento
3	Supervisor
4	Analista de Sistemas
5	Analista de Negócios

Funcionario

Cod_Func Chave Primária	Cod_Cargo Chave Estrangeira	Nome_Func	Sal_Func	RG_Func
1	5	José Carlos	4500.00	15.253.587
2	5	Pedro Cardoso	4500.00	14.258.457
3	3	Antonieta Antonino	3500.00	17.258.368
4	3	Amélia Pedreira	5500.00	18.963.458
5	2	Cristina Aparecida	6000.00	16.258.147
6	1	Tereza Tibiriçá	10000.00	14.235.258

Observemos que a coluna Cod_Cargo existe nas duas tabelas. Na tabela Cargo, essa coluna é a chave primária, e, na tabela Funcionário, ela é a chave estrangeira. Se precisarmos saber o nome de cada cargo e o nome de cada funcionário que existe em cada cargo, teremos que associar as duas tabelas por meio dessa coluna.

Para conseguirmos enxergar melhor os dados desejados, observemos a tabela a seguir:

Nome_Cargo	Nome_Func
Analista de Negócios	José Carlos Gonçalves
Analista de Negócios	Pauto Cardos»
Supervisor	Antonieta António
Supervisor	Amélia Pedreira
Diretor de Departamento	Cristina Aparecida
Diretor Presidente	Tereza Tibiriçá

Observemos que a tabela anterior mostra apenas os dados relacionados, ou seja, os cargos que possuem funcionários e os funcionários que possuem cargo. O cargo Analista de Sistemas, por não ter nenhum funcionário, não aparece na lista anterior.

Para fazer com que o SQL Server exiba os dados relacionados dessa forma, executamos o comando seguinte;

Com a cláusula **INNER JOIN**

```
SELECT Cargo. Nome_Cargo, Funcionario.Nome_Func
FROM Cargo
INNER JOIN Funcionario
ON Cargo. Cod_Cargo = Funcionario.Cod_Cargo
```

Com a cláusula **WHERE**

```
SELECT Cargo. Nome_Cargo, Funcionario.Nome_Func
FROM Cargo, Funcionario
WHERE Cargo. Cod_Cargo = Funcionario.Cod_Cargo
```

Observemos que, nos dois tipos de associação, as tabelas foram relacionadas pela chave primária da tabela Cargo e estrangeira da tabela Funcionário.

OBS: Apesar da cláusula **INNER** ser opcional, há um bug no MySQL em que ocorre erro ao ser omitido. Para evitar este tipo de erro, declare a cláusula **INNER** explicitamente.

7.2 - Obtendo os dados relacionados e não- relacionados da tabela do lado esquerdo – LEFT JOIN

Para obtermos os dados relacionados entre as duas tabelas e os não-relacionados que estiverem na tabela do lado esquerdo da cláusula JOIN. Como mostra a tabela seguinte, utilizamos a cláusula Left Join.

Observe:

Com a cláusula **LEFT JOIN**:

```
SELECT Cargo. Nome_Cargo, Funcionario.Nome_Func  
FROM Cargo  
LEFT OUTER JOIN Funcionario  
ON Cargo.Cod_Cargo = Funcionario.Cod_Cargo
```

7.3 - Obtendo os dados relacionados e não-relacionados da tabela do lado direito – RIGHT JOIN

Se invertermos a posição das tabelas nos comandos anteriores, teremos de usar as cláusulas RIGHT OUTER JOIN para obtermos os mesmos dados.

- Com a cláusula **RIGHT JOIN**:

```
SELECT Cargo. Nome_Cargo, Funcionario. Nome_Func  
FROM Funcionario  
RIGHT OUTER JOIN Cargo  
ON Funcionario.Cod_Cargo = Cargo.Cod_Cargo
```

Suponhamos que nas tabelas seguintes existam exatamente esses dados, mas que não haja nenhuma chave primária, nem chave estrangeira. Sendo assim, imaginemos que algum usuário tenha inserido um funcionário para um cargo inexistente na tabela Cargo.

Cargo

Cod_Cargo	Nome_Cargo Chave Única
1	Diretor Presidente
2	Diretor de Departamento
3	Supervisor
4	Analista de Sistemas
5	Analista de Negócios

Funcionario

Cod_Func Chave Primária	Cod_Cargo	Nome_Func	Sal_Func	RG_Func
1	5	José Carlos	4500.00	15.253.587
2	5	Pedro Cardoso	4500.00	14.258.457
3	3	Antonieta Antonino	3500.00	17.258.368
4	3	Amélia Pedreira	5500.00	18.963.458
5	2	Cristina Aparecida	6000.00	16.258.147
6	1	Tereza Tibiriçá	10000.00	14.235.258
7	1000	Débora Denito	2500.00	17.782.256

Se quisermos exibir todos os funcionários e seus cargos, mesmo que o cargo do funcionário não exista na tabela Cargo, escrevemos o comando a seguir:

Com a cláusula RIGHT JOIN

```
SELECT Cargo. Nome_Cargo, Funcionario.Nome_Func
FROM Cargo
RIGHT JOIN Funcionario
ON Cargo. Cod_Cargo = Funcionario.Cod_Cargo
```

Se invertermos as tabelas de posição podemos escrever Left Join no lugar de Right Join .

Observe:

Com a cláusula LEFT JOIN

```
SELECT Cargo. Nome_Cargo, Funcionario.Nome_Func
FROM Funcionario
LEFT JOIN Cargo
ON Funcionario.Cod_Cargo = Cargo.Cod_Cargo
```

Obs: Poderá ser utilizado nas cláusulas Right Join e Left Join a palavra reservada OUTER, ficando Right Outer Join ou Left Outer Join

Nome_Cargo	Nome_Func
Analista de Negócios	José Carlos Gonçalves
Analista de Negócios	Pauto Cardos»
Supervisor	Antonieta António
Supervisor	Amélia Pedreira
Diretor de Departamento	Cristina Aparecida
Diretor Presidente	Tereza Tibiriçá
	Débora Denito

7.4 - Associando mais de duas tabelas

Para associarmos mais de duas tabelas, será utilizado o modelo a seguir:

Cargo

Cod_Cargo	Nome_Cargo Chave Única
1	Diretor Presidente
2	Diretor de Departamento
3	Supervisor
4	Analista de Sistemas
5	Analista de Negócios

Funcionario

Cod_Func Chave Primária	Cod_Cargo	Nome_Func	Sal_Func	RG_Func
1	5	José Carlos	4500.00	15.253.587
2	5	Pedro Cardoso	4500.00	14.258.457
3	3	Antonieta Antonino	3500.00	17.258.368
4	3	Amélia Pedreira	5500.00	18.963.458
5	2	Cristina Aparecida	6000.00	16.258.147
6	1	Tereza Tibiriçá	10000.00	14.235.258
7	1000	Débora Denito	2500.00	17.782.256

Dependente

Cod_Dep Chave Primária	Cod_Func Chave Estrangeira	Nome_Dep
1	1	Rosalina Reguete
2	1	Marcelina Reguete
3	1	Claudio Mirol Reguete
4	2	Amarildo Cardoso
5	3	Anacleto Cardoso
6	5	Paula Aparecida
7	5	Natalia Aparecida

Caso1

Para obtermos apenas os dados relacionados:

Nota: Todos os funcionários possuem cargo, mas nem todo funcionário possui dependente.

Escrevemos um dos comandos a seguir:

Primeira forma:

```
SELECT Funcionario.Nome_Func, Cargo.Nome_Cargo,
       Dependente.Nome_Dep
FROM Cargo
INNER JOIN Funcionario
    ON Cargo.Cod_Cargo = Funcionario.Cod_Cargo
INNER JOIN Dependente
    ON Funcionario.Cod_Func = Dependente.Cod_Func
```

Segunda forma:

```
SELECT Funcionario.Nome_Func, Cargo.Nome_Cargo,
       Dependente.Nome_Dep
FROM Cargo, Funcionario, Dependente
WHERE Cargo.Cod_Cargo = Funcionario.Cod_Cargo
AND Funcionario.Cod_Func = Dependente.Cod_Func
```

Caso 2

Para obtermos funcionários e seus cargos, inclusive os funcionários que não têm dependentes:

```
SELECT Funcionario.Nome_Func, Cargo.Nome_Cargo,
       Dependente.Nome_Dep
FROM Cargo
INNER JOIN Funcionario
    ON Cargo.Cod_Cargo = Funcionario.Cod_Cargo
LEFT JOIN Dependente
    ON Funcionario.Cod_Func = Dependente.Cod_Func
```


Caso 3

Para obtermos todos os cargos com e sem funcionários e os funcionários com e sem dependentes:

```
SELECT Funcionario.Nome_Func, Cargo.Nome_Cargo,
       Dependente.Nome_Dep
FROM Cargo
LEFT JOIN Funcionario
      ON Cargo.Cod_Cargo = Funcionario.Cod_Cargo
LEFT JOIN Dependente
      ON Funcionario.Cod_Func = Dependente.Cod_Func
```

7.5 - CROSS JOIN

Quando queremos cruzar os dados de duas ou mais tabelas, utilizamos o **CROSS JOIN**.

Para expormos esse assunto, utilizaremos o exemplo das tabelas relacionadas Aluno, Materia e AlunoMat.

Usando a cláusula **CROSS JOIN** é possível cruzarmos os dados de Alunos e de Matérias, fazendo com que, na exibição dos mesmos, todos os alunos cursem todas as matérias e todas as matérias sejam cursadas por todos os alunos:

Primeira Forma:

```
SELECT Aluno.Nome_Aluno, Materia.Nome_Mat
FROM Aluno CROSS JOIN Materia
ORDER BY 1
```

Segunda Forma:

```
SELECT Aluno.Nome_Aluno, Materia.Nome_Mat
FROM Aluno, Materia
ORDER BY 1
```

7.6 - Update e Delete com JOIN

Se for necessário alterar ou excluir dados de uma tabela A com base nos dados de uma tabela B, se as tabelas A e B se relacionam entre si, será possível a utilização da cláusula JOIN junto com os comandos UPDATE ou DELETE.

Observemos o exemplo a seguir, que utiliza as tabelas Funcionário e Pedido como exemplo:

Funcionario

Cod_Func	Nome_Func	Sal_Func
1	Ricardo Oliveira	1500.00
2	Marcos de Oliveira	2000.00
3	Márcio Garcia	3000.00
4	Cândido Claro	2500.00

Pedido

Num_Ped	Cod_Func	Data_Ped	Val_Ped
1	1	01/02/03	200.00
2	1	02/03/03	300.00
3	2	02/03/03	400.00
4	3	03/03/03	500.00

7.7 - Update com JOIN

Para exemplificar, suponha que seja necessário atualizarmos todos os pedidos dos funcionários que recebem um salário de 1500.00, dando um desconto de 10%.

```
UPDATE PEDIDO
  SET VAL_PED = VAL_PED * 0.9
  FROM FUNCIONARIO
  INNER JOIN PEDIDO
    ON FUNCIONARIO.COD_FUNC = PEDIDO.COD_FUNC
  WHERE SAL_FUNC = 1500
```

7.8 - Delete com JOIN

O comando DELETE também pode ser utilizado com a cláusula JOIN. Para exemplificar, suponha que seja necessário excluirmos todos os pedidos dos funcionários que recebem um salário de 1500.00.

Os pedidos dos funcionários que recebem um salário de 1500.00 são estes:

```
SELECT Pedido.*, Funcionario.Sal_Func
  FROM Funcionario
  INNER JOIN Pedido
    ON Funcionario.Cod_Func = Pedido.Cod_Func
  WHERE Funcionario.Sal_Func = 1500.00
```

Para excluir os dados em questão, escrevemos o comando DELETE a seguir com a cláusula FROM do comando SELECT anterior.

```
DELETE Pedido
FROM Funcionario
INNER JOIN Pedido
    ON Funcionario.Cod_Func = Pedido.Cod_Func
WHERE Funcionario.Sal_Func = 1500.00
```

Exercícios (Utilizar database DB_CDS)

- 1 - Selecionar o nome dos CDs e o nome da gravadora de cada CD;
2. Selecionar o nome dos CDs e o nome de cada categoria dos CDs;
3. Selecionar o nome dos CDs, o nome da gravadora de cada CD e o nome da categoria de cada CD;
4. Selecionar o nome dos clientes e os títulos dos CDs vendidos em cada pedido que o cliente fez;
5. Selecionar o nome do funcionário, número, data e valor dos pedidos que esse funcionário registrou, e o nome do cliente que está fazendo esse pedido;
6. Selecionar o nome dos funcionários e o nome de todos os dependentes que cada funcionário possui;
7. Selecionar o nome dos clientes e o nome dos cônjuges de cada cliente;
8. Selecionar o nome de todos os clientes. Se estes tiverem cônjuges, mostrar os nomes de seus cônjuges também;
9. Selecionar:
 - Nome do cliente;
 - Nome do cônjuge;
 - Número do pedido que cada cliente fez;
 - Valor do pedido que cada cliente fez;
 - Código do funcionário que atendeu cada pedido.

8 – Cláusula UNION ALL

A cláusula **UNION ALL** é utilizada para obter dados de mais de uma tabela, sem associá-las, mas unindo os resultados de mais de um **SELECT**.

Para tanto, observemos as três tabelas a seguir: Cliente, Funcionário e Aluno. Notemos que essas tabelas não estão relacionadas entre si, portanto, se for preciso obtermos um relatório de todas as pessoas existentes no database, será necessária a utilização da cláusula **UNION**.

As tabelas e seus dados:

Cliente

COD_CLI	NOME_CLI	SEXO_CLI	SAL_CLI
1	Ana de Souza	F	1500.23
2	Mariana de Souza	F	2800.15
3	Fabiana de Souza	F	287.56
4	Rosana de Souza	F	5589.58
5	Adriana de Souza	F	6785.59

Funcionario

COD_FUNC	NOME_FUNC	SAL_FUNC
1	Paulino Reguete	4500.00
2	Romeu Pompilho	4500.00
3	Antonieta Antonino	3500.00
4	Amélia Amália	5500.00
5	Crisântemo Cristalino	6000.00
6	Amim Amou Amado	10000.00

Aluno

COD_ALUNO	NOME_ALUNO	SEXO_ALUNO
1	Pompeu de Tibiriçá	M
2	Amaro Amarildo	M
3	Márcia Mirtes	F

Criando as tabelas e inserindo os dados:

```
-- CRIANDO O DATABASE
CREATE DATABASE UNION_ALL
-- USANDO O DATABASE
USE UNION_ALL
```

```
-- CRIANDO AS TABELAS
```

```
CREATE TABLE CLIENTE
```

```
(
  COD_CLI      INT              NOT NULL,
  NOME_CLI     CHAR(50)         NOT NULL,
  SEXO_CLI     CHAR(1)          NOT NULL,
  SAL_CLI      DECIMAL(10,2)    NOT NULL,
  CONSTRAINT PK_CLIENTE PRIMARY KEY(COD_CLI)
);
```

```
CREATE TABLE FUNCIONARIO
```

```
(
  COD_FUNC     INT              NOT NULL,
  NOME_FUNC    CHAR(50)         NOT NULL,
  SAL_FUNC     DECIMAL(10,2)    NOT NULL,
  CONSTRAINT PK_FUNC PRIMARY KEY (COD_FUNC)
);
```

```
CREATE TABLE ALUNO
```

```
(
  COD_ALUNO    INT              NOT NULL,
  NOME_ALUNO   CHAR(50)         NOT NULL,
  SEXO_ALUNO   CHAR(1)          NOT NULL,
  CONSTRAINT PK_ALUNO PRIMARY KEY (COD_ALUNO)
);
```

```
-- INSERINDO DADOS NA TABELA
```

```
INSERT CLIENTE VALUES (1,'Ana de Souza','F', 1500.23)
INSERT CLIENTE VALUES (2,'Ana de Souza','F', 2800.15)
INSERT CLIENTE VALUES (3,'Ana de Souza','F', 287.56)
INSERT CLIENTE VALUES (4,'Ana de Souza','F', 5589.58)
INSERT CLIENTE VALUES (5,'Ana de Souza','F', 6785.59)
```

```
INSERT INTO FUNCIONARIO VALUES (1,'Anatalino Reguete',4500.00)
INSERT INTO FUNCIONARIO VALUES (2,'Romeu Pompilho',4500.00)
INSERT INTO FUNCIONARIO VALUES
      (3,'Antonietta Antonino',3500.00)
INSERT INTO FUNCIONARIO VALUES (4,'Amélia Amália',5500.00)
INSERT INTO FUNCIONARIO VALUES
      (5,'Crisântemo Cristalino',6000.00)
INSERT INTO FUNCIONARIO VALUES (6,'Amim Amou Amado',10000.00)
```

```
INSERT INTO ALUNO  VALUES (1,'Pompeu de Tibiriça','M')
INSERT INTO ALUNO  VALUES (2,'Amaro Amarildo','M')
INSERT INTO ALUNO  VALUES (3,'Marcia Mirtes','F')
```

Dados desejados

1	ALUNO	Pompeu de Tibiriça	M	NULL
2	ALUNO	Amaro Amarildo	M	NULL
3	ALUNO	Marcia Mirtes	F	NULL
1	CLIENTE	Ana de Souza	F	1500.23
2	CLIENTE	Mariana de Souza	F	2800.15
3	CLIENTE	Fabiana de Souza	F	287.56
4	CLIENTE	Rosana de Souza	F	5589.58
5	CLIENTE	Adriana de Souza	F	6785.59
1	FUNCIONARIO	Anatalino Reguete	EM BRANCO	4500.00
2	FUNCIONARIO	Romeu Pompilho	EM BRANCO	4500.00
3	FUNCIONARIO	Antonietta Antonino	EM BRANCO	3500.00
4	FUNCIONARIO	Amélia Amália	EM BRANCO	5500.00
5	FUNCIONARIO	Crisântemo Cristalino	EM BRANCO	6000.00
6	FUNCIONARIO	Amim Amou Amado	EM BRANCO	10000.00

Observamos que a tabela **Cliente** contém informações de código, nome, sexo e salário, a tabela **Funcionario** não possui a coluna sexo e a tabela **Aluno** não possui a coluna salário. Porém, se for necessário obtermos os dados das três tabelas apresentando as colunas código, tipo, nome, sexo e salário, como mostra a tabela anterior, será preciso escrever três comandos SELECT e unir com a cláusula UNION os resultados dos mesmos.

Observemos:

```

SELECT COD_CLI    AS COD_PES, 'CLIENTE' AS TIPO_PES,
       NOME_CLI   AS NOME_PES, SEXO_CLI  AS SEXO_PES,
       SAL_CLI    AS SAL_PES
FROM CLIENTE

UNION ALL

SELECT COD_FUNC, 'FUNCIONARIO',
       NOME_FUNC, 'EM BRANCO',
       SAL_FUNC
FROM FUNCIONARIO

UNION ALL

SELECT COD_ALUNO, 'ALUNO',
       NOME_ALUNO, SEXO_ALUNO,
       NULL
FROM ALUNO

ORDER BY 2,1

```

8.1 Regras para utilização da cláusula **UNION ALL**

Para a utilização da cláusula **UNION ALL**, existem algumas regras que devem ser seguidas:

- Todos os comandos **SELECT** envolvidos no processo deverão apresentar a mesma quantidade de colunas;
- Todos os comandos **SELECT** deverão apresentar as colunas na mesma sequência de datatypes. Por exemplo, se a coluna 1 do primeiro **SELECT** for do datatype **int**, a primeira coluna do segundo **SELECT** também deverá ser do datatype **int**. O mesmo acontece com a primeira coluna dos outros comandos **SELECT** envolvidos nesse processo. Este procedimento também deverá ser seguido para as segundas, terceiras e demais colunas de todos os comandos **SELECT**;
- Se o nome (alias) atribuído às colunas for utilizado, este deverá ser acrescentado no primeiro **SELECT**;
- Qualquer um dos comandos **SELECT** poderá ter a sua cláusula **WHERE**;
- Qualquer **SELECT** poderá ser escrito com **Join** ou **subquery**, se necessário;
- Se for preciso ordenar os dados, a cláusula **ORDER BY** deverá ser escrita após o último **SELECT**.

A cláusula **UNION** (sem a cláusula **ALL**) mostrará apenas uma vez os dados que estiverem repetidos em mais de uma tabela.

9 – Subquery

As subqueries permitem que seja feita uma pesquisa nos dados de uma tabela com base na existência ou não nos dados de outra tabela.

Para expor esse assunto, serão utilizadas as tabelas com os dados a seguir:

Cargo

Cod_Cargo Chave Primária	Nome_Cargo Chave Única
1	Diretor Presidente
2	Diretor de Departamento
3	Supervisor
4	Analista de Sistemas
5	Analista de Negócios

Funcionario

Cod_Func Chave Primária	Cod_Cargo Chave Estrangeira	Nome_Func	Sal_Func	RG_Func
1	5	Pauliino Requete	4500.00	15.253.587
2	5	Romeu Pompilho	4500.00	14.258.457
3	3	Antonieta Antonino	3500.00	17.258.368
4	3	Amélia Amália	5500.00	18.963.458
5	2	Crisântemo Cristalino	6000.00	16.258.147
6	1	Amim Amou Amado	10000.00	14.235.258

Dependente

Cod_Dep Chave Primária	Cod_Func Chave Estrangeira	Nome_Dep
1	1	Rosalina Reguete
2	1	Marcelina Reguete
3	1	Claudio Reguete
4	2	Amarildo Pomposo
5	3	Anacleto Pomposo
6	5	Pepino Crisântemo
7	5	Natalino Crisântemo

• Criando as tabelas e inserindo os dados

```
-- CRIANDO DATABASE
CREATE DATABASE SUBQUERY;
```

```
-- USANDO DATABASE
USE SUBQUERY;
-- CRIANDO AS TABELAS
```



```
CREATE TABLE CARGO
```

```
(
    COD_CARGO INT NOT NULL,
    NOME_CARGO CHAR(30) NOT NULL,
    CONSTRAINT PK_CARGO PRIMARY KEY (COD_CARGO),
    CONSTRAINT UQ_CARGO UNIQUE (NOME_CARGO)
)
```

```
CREATE TABLE FUNCIONARIO
```

```
(
    COD_FUNC INT NOT NULL,
    COD_CARGO INT NOT NULL,
    NOME_FUNC CHAR(50) NOT NULL,
    SAL_FUNC DECIMAL(10,2) NOT NULL,
    RG_FUNC CHAR(15),
    CONSTRAINT PK_FUNC PRIMARY KEY(COD_FUNC),
    CONSTRAINT UQ_FUNC UNIQUE (RG_FUNC),
    CONSTRAINT FK_FUNC FOREIGN KEY (COD_CARGO)
        REFERENCES CARGO (COD_CARGO)
)
```

```
CREATE TABLE DEPENDENTE
```

```
(
    COD_DEP INT NOT NULL,
    COD_FUNC INT NOT NULL,
    NOME_DEP CHAR(50) NOT NULL,
    CONSTRAINT PK_DEP PRIMARY KEY (COD_DEP),
    CONSTRAINT FK_DEP FOREIGN KEY (COD_FUNC)
        REFERENCES FUNCIONARIO (COD_FUNC)
)
```

```
-- INSERINDO DADOS
```

```
INSERT CARGO VALUES (1, 'Diretor Presidente')
```

```
INSERT CARGO VALUES (2, 'Diretor de Departamento')
```

```
INSERT CARGO VALUES (3, 'Supervisor')
```

```
INSERT CARGO VALUES (4, 'Analista de Sistemas')
```

```
INSERT CARGO VALUES (5, 'Analista de Negócios')
```

```
INSERT FUNCIONARIO VALUES
```

```
(1,5,'Paulino Reguete',4500.00,'15.253.587')
```

```
INSERT FUNCIONARIO VALUES
```

```
(2,5,'Romeu Pompilho',4500.00,'14.258.457')
```

```
INSERT FUNCIONARIO VALUES
```

```
(3,3,'Antonietta Antonio',3500.00,'17.258.368')
```

```
INSERT FUNCIONARIO VALUES
```

```
(4,3,'Amélia Amália',5500.00,'18.963.458')
```

```
INSERT FUNCIONARIO VALUES
```

```
(5,2,'Crisântemo Cristalino',6000.00,'16.258.147')
```

```
INSERT FUNCIONARIO VALUES
```

```
(6,1,'Amim Amou Amado',10000.00,'14.235.258')
```

```
INSERT DEPENDENTE VALUES (1,1,'Rosalina Reguete')
INSERT DEPENDENTE VALUES (2,1,'Marcelina Reguete')
INSERT DEPENDENTE VALUES (3,1,'Claudomiro Reguete')
INSERT DEPENDENTE VALUES (4,2,'Amarildo Pomposo')
INSERT DEPENDENTE VALUES (5,2,'Anacleto Pomposo')
INSERT DEPENDENTE VALUES (6,5,'Pepino Crisântemo')
INSERT DEPENDENTE VALUES (7,5,'Natalino Crisântemo')
```

9.1 – Subqueries introduzidas com IN/NOT

Suponhamos que seja necessário obter da tabela Cargo todos os cargos que estão sem nenhum funcionário.

Observando as tabelas Cargo e Funcionário, é fácil percebermos que apenas o cargo de número 4 está sem nenhum funcionário, porque seu código não existe na tabela Funcionário. Senão assim, para fazer com que o MySQL retorne essa informação, poderemos escrever o comando seguinte:

```
SELECT * FROM CARGO
WHERE COD_CARGO NOT IN
      (SELECT COD_CARGO FROM FUNCIONARIO)
```

O MSSQL executará o segundo **SELECT** do comando anterior e colocará esses dados na memória. Em seguida, o MSSQL executará o outro **SELECT** mais externo e retornará apenas os dados que não existirem no conjunto de dados da memória, sendo, neste caso, apenas o cargo 4, Analista de Sistemas.

A resposta obtida será:

Cargo

Cod_Cargo	Nome_Cargo
4	Analista de Sistemas

Por outro lado, se fosse necessário obtermos como resposta apenas os cargos para os quais já foi cadastrado pelo menos um funcionário, em vez de introduzirmos a subquery com os operadores NOT IN, utilizaríamos apenas IN.

Observemos:

```
SELECT * FROM CARGO
WHERE COD_CARGO IN
      (SELECT COD_CARGO FROM FUNCIONARIO)
```

A resposta obtida seria:

Cargo

Cod_Cargo Chave Primária	Nome_Cargo Chave Única
1	Diretor Presidente
2	Diretor de Departamento
3	Supervisor
5	Analista de Negócios

Imaginemos agora que seja necessário obtermos os cargos dos funcionários que têm dependentes. Utilizando subquery, esse comando ficaria assim:

```
SELECT * FROM CARGO
WHERE COD_CARGO IN
  (SELECT COD_CARGO FROM FUNCIONARIO
   WHERE COD_FUNC IN
     (SELECT COD_FUNC FROM DEPENDENTE))
```

A resposta seria:

Cargo

Cod_Cargo Chave Primária	Nome_Cargo Chave Única
2	Diretor de Departamento
5	Analista de Negócios

Se, por sua vez, fosse necessário obtermos os cargos dos funcionários que não têm dependentes, o comando a ser executado seria este:

```
SELECT * FROM CARGO
WHERE COD_CARGO IN
  (SELECT COD_CARGO FROM FUNCIONARIO
   WHERE COD_FUNC NOT IN
     (SELECT COD_FUNC FROM DEPENDENTE))
```

A resposta obtida seria:

Cargo

Cod_Cargo Chave Primária	Nome_Cargo Chave Única
1	Diretor Presidente
3	Supervisor

9.2 – Subqueries introduzidas com o sinal de =

Observando os dados da tabela a seguir, podemos perceber que o funcionário que tem o salário mais alto é o de código 6 e o funcionário que tem o salário mais baixo é o de código 3.

Para fazermos com que o MySQL retorne essa informação, teremos que utilizar as funções de totalização Max() e Min().

A função **Max()** nos dará o maior valor de um determinado conjunto e a função **Min()** o menor.

Sendo assim, para que sejam exibidos todos os dados do funcionário que tem o maior salário, escrevemos a query seguinte:

```
SELECT * FROM FUNCIONARIO
WHERE SAL_FUNC =
      (SELECT MAX(SAL_FUNC) FROM FUNCIONARIO)
```

Nesse caso, o MSSQL executará a subquery e obterá o valor 10000.00, colocando-o na memória. Em seguida, executará o SELECT mais externo e obterá todos os funcionários cujo salário seja igual ao valor colocado anteriormente na memória.

A resposta obtida será:

Funcionario

Cod_Func	Cod_Cargo	Nome_Func	Sal_Func	RG_Func
6	1	Amim Amou Amado	10000.00	14.235.258

Para obtermos o funcionário que tem o menor salário, escrevemos o comando a seguir:

```
SELECT * FROM FUNCIONARIO
WHERE SAL_FUNC =
      (SELECT MIN(SAL_FUNC) FROM FUNCIONARIO)
```

A resposta obtida será:

Funcionario

Cod_Func	Cod_Cargo	Nome_Func	Sal_Func	RG_Func
3	3	Antonieta Antonino	3500.00	17.258.368

Dentre os funcionários que possuem dependentes, selecionemos aqueles que têm o menor salário. Para obtermos esses dados utilizando subquery, será necessário escrevermos o seguinte comando:

```

SELECT * FROM FUNCIONARIO
WHERE SAL_FUNC =
      (SELECT MIN(SAL_FUNC) FROM FUNCIONARIO
       WHERE COD_FUNC IN
        (SELECT COD_FUNC FROM DEPENDENTE))

```

A resposta obtida será:

Funcionario

Cod_Func	Cod_Cargo	Nome_Func	Sal_Func	RG_Func
1	5	Pauliino Requete	4500.00	15.253.587
2	5	Romeu Pompilho	4500.00	14.258.457

De acordo com os quadros a seguir, vamos exibir todos os dados de cada funcionário e a quantidade de dependentes que cada funcionário possui, utilizando subqueries.

Funcionario

Cod_Func	Cod_Cargo	Nome_Func	Sal_Func	RG_Func
1	5	Pauliino Requete	4500.00	15.253.587
2	5	Romeu Pompilho	4500.00	14.258.457
3	3	Antonieta Antonino	3500.00	17.258.368
4	3	Amélia Amália	5500.00	18.963.458
5	2	Crisântemo Cristalino	6000.00	16.258.147
6	1	Amim Amou Amado	10000.00	14.235.258

Dependente

Cod_Dep	Cod_Func	Nome_Dep
1	1	Rosalina Requete
2	1	Marcelina Requete
3	1	Claudio Miros Requete
4	2	Amarildo Pomposo
5	3	Anacleto Pomposo
6	5	Pepino Crisântemo
7	5	Natalino Crisântemo

Para obtermos os dados desejados, será necessário escrevermos uma subquery correlacionada, Esta ficará na lista do **SELECT**, no lugar de uma coluna.

Observemos:

```

SELECT *,
      (SELECT COUNT(*) FROM DEPENDENTE
       WHERE COD_FUNC = FUNCIONARIO.COD_FUNC)
      As QTD_DEP
FROM FUNCIONARIO

```

A subquery está correlacionada com a query externa por meio da tabela funcionário.

A resposta obtida será:

Funcionario

Cod_Func	Cod_Cargo	Nome_Func	Sal_Func	RG_Func	Qtd_Dep
1	5	Pauliino Requete	4500.00	15.253.587	3
2	5	Romeu Pompilho	4500.00	14.258.457	2
3	3	Antonieta Antonino	3500.00	17.258.368	0
4	3	Amélia Amália	5500.00	18.963.458	0
5	2	Crisântemo Cristalino	6000.00	16.258.147	2
6	1	Amim Amou Amado	10000.00	14.235.258	0

Para obtermos código e nome dos cargos e a quantidade de funcionários que cada cargo possui, será necessário escrevermos uma query semelhante.

Observemos:

```
SELECT *,
      (SELECT Count(*) FROM FUNCIONARIO
       WHERE COD_CARGO = CARGO.COD_CARGO)
      As QTD_FUNC FROM CARGO
```

A subquery está correlacionada com a query externa por meio da tabela Cargo. A função Count() é utilizada para contar a quantidade de linhas existentes em uma tabela.

A resposta obtida será:

Cargo

Cod_Cargo	Nome_Cargo	Qtd_Func
1	Diretor Presidente	2
2	Diretor de Departamento	0
3	Supervisor	1
4	Analista de Sistemas	1
5	Analista de Negócios	2

9.3 – Regras das subqueries

- Toda subquery deve ser escrita entre parênteses;
- Toda subquery só poderá retornar uma coluna;
- As subqueries introduzidas com o sinal de igual (=) só poderão retornar um valor.

9.4 – Update com subqueries

Vamos aplicar 10% de aumento salarial apenas para todos os funcionários que não têm dependentes. Para aplicarmos esse aumento salarial utilizando subquery, executamos o comando a seguir:

```
UPDATE FUNCIONARIO
  SET SAL_FUNC = SAL_FUNC * 1.1
  WHERE COD_FUNC NOT IN
        (SELECT COD_FUNC FROM DEPENDENTE)
```

9.5 – Delete com subqueries

Vamos excluir todos os funcionários que não têm dependentes. Para realizarmos essa tarefa, executamos o comando seguinte:

```
DELETE FROM FUNCIONARIO
  WHERE COD_FUNC NOT IN
        (SELECT COD_FUNC FROM DEPENDENTE)
```

Exercícios:

Utilizando o database DB_CDS, faça os exercícios abaixo:

- 1 – Apresentar todos os clientes solteiros;
- 2 – Apresentar todos os clientes casados;
- 3 – Apresentar todas as categorias que estão sem CD;
- 4 – Apresentar todas as gravadoras que nunca gravaram CD;
- 5 – Apresentar apenas os funcionários que tem dependentes e que nunca atenderam a algum pedido;
- 6 – Mostrar todos os funcionários que atenderam a pedidos de clientes casados;
- 7 – Apresentar os dados dos clientes que possuem a maior renda;
- 8 – Apresentar os dados do CD mais caro.

Exercícios – Parte II

1. Aplicar um aumento salarial de 20% (* 1.2) para todos os funcionários que não têm dependentes;
2. Aplicar um aumento salarial; de 10% (* 1.1). para todos os funcionários que têm dependentes;
3. Aumentar em 100.00 a renda de todos os clientes solteiros;
4. Aumentar em 100.00 a renda de todos os clientes solteiros do sexo feminino;
5. Aumentar em 200.00 a renda de todos os clientes que já fizeram pedidos;
6. Aumentar em 500.00 a renda de todos os clientes solteiros que já fizeram pedidos;
7. Aplicar um desconto de 10% (*0.9) em todos os pedidos onde foi vendido o título de código igual a 1;
8. Aumentar em 500.00 o salário de todos os funcionários que já atenderam a pedidos feitos pelo cliente 1;
9. Aplicar um desconto de 10% (*0.9) para todos os CDs (títulos) que nunca foram vendidos;
10. Aplicar um aumento salarial de 20% (*1.2) para todos os funcionários que não têm dependentes.

10 – Cláusula GROUP BY

A cláusula GROUP BY é utilizada para agrupar dados de maneira que estes possam ser totalizados.

Junto com GROUP BY, utilizamos as funções:

- SUM() para realizar somas;
- COUNT() para realizar contagens de registros;
- MAX() para obter o maior valor;
- MIN() para obter o menor valor;
- AVG() para calcular médias .

Para exemplificar a utilização dessa cláusula, vamos utilizar a tabela **Produto** a seguir:

Produto

Cod_Prod	Tipo_Prod	Linha_Prod	Nome_Prod	Qtd_Prod	Val_Prod
1	A	1	Mesa	100	500.00
2	B	1	Sofá	200	1500.00
3	A	2	Mesa	100	200.00
4	A	2	Armário	300	200.00
5	C	1	Cama	500	1000.00
6	B	1	Poltrona	100	250.00
7	B	2	Sofá	200	300.00
8	C	2	Cama	100	150.00
9	A	1	Armário	200	800.00
10	C	1	Guarda_Roupa	100	1500.00

Criando as tabelas e inserindo os dados:

```
-- CRIANDO O DATABASE
CREATE DATABASE GROUP_BY
GO
-- USANDO O DATABASE
USE GROUP_BY
GO
-- CRIANDO AS TABELA
CREATE TABLE LINHA
(
    LINHA_PROD TINYINT NOT NULL,
    NOME_LINHA CHAR(50) NOT NULL,
    CONSTRAINT PK_LINHA PRIMARY KEY (LINHA_PROD),
    CONSTRAINT UQ_LINHA UNIQUE (NOME_LINHA)
)
GO
```

```
CREATE TABLE TIPOPROD
(
    TIPO_PROD      CHAR(1)          NOT NULL,
    NOME_TIPO      VARCHAR(50)      NOT NULL,
    CONSTRAINT PK_TIPOPROD PRIMARY KEY(TIPO_PROD),
    CONSTRAINT UQ_TIPOPROD UNIQUE (NOME_TIPO)
)
GO
CREATE TABLE PRODUTO
(
    COD_PROD       INT              NOT NULL,
    TIPO_PROD      CHAR(1)          NOT NULL,
    LINHA_PROD     TINYINT          NOT NULL,
    NOME_PROD      VARCHAR(50)      NOT NULL,
    QTD_PROD       INT              NOT NULL,
    VAL_PROD       DECIMAL(10,2)    NOT NULL,
    CONSTRAINT PK_PROD PRIMARY KEY (COD_PROD),
    CONSTRAINT FK1_PROD FOREIGN KEY (TIPO_PROD)
        REFERENCES TIPOPROD(TIPO_PROD),
    CONSTRAINT FK2_PROD FOREIGN KEY(LINHA_PROD)
        REFERENCES LINHA (LINHA_PROD)
)
GO
-- INSERINDO DADOS
INSERT LINHA VALUES (1,'Primeira Linha')
INSERT LINHA VALUES (2,'Segunda Linha')
INSERT LINHA VALUES (3,'Terceira Linha')

INSERT TIPOPROD VALUES ('A','Cozinha')
INSERT TIPOPROD VALUES ('B','Sala')
INSERT TIPOPROD VALUES ('C','Quarto')

INSERT PRODUTO VALUES (1,'A',1,'Mesa',100,500.00)
INSERT PRODUTO VALUES (2,'B',1,'Sofá',200,1500.00)
INSERT PRODUTO VALUES (3,'A',2,'Mesa',100,200.00)
INSERT PRODUTO VALUES (4,'A',2,'Armário',300,200.00)
INSERT PRODUTO VALUES (5,'C',1,'Cama',500,1000.00)
INSERT PRODUTO VALUES (6,'B',1,'Poltrona',100,250.00)
INSERT PRODUTO VALUES (7,'B',2,'Sofá',200,300.00)
INSERT PRODUTO VALUES (8,'C',2,'Cama',100,150.00)
INSERT PRODUTO VALUES (9,'A',1,'Armário',200,800.00)
INSERT PRODUTO VALUES
    (10,'C',1,'Guarda-Roupa',100,1500.00)
```

No primeiro exemplo da utilização da cláusula GROUP BY, suponhamos que seja necessário exibir uma somatória da quantidade de produtos em estoque, separando os produtos por tipo, da seguinte forma:

Tipo_Prod	Qtd_Estq
A	700
B	500
C	700

Para exibir esse resultado, deveremos agrupar os produtos por tipo e, em seguida, somar a quantidade em estoque de cada um deles. Observemos o comando a seguir:

```
SELECT TIPO_PROD, SUM(QTD_PROD) AS QTD_ESTQ
FROM PRODUTO
GROUP BY TIPO_PROD
```

Em um segundo exemplo da utilização da cláusula GROUP BY, suponhamos que seja necessário exibir uma somatória da quantidade de produtos em estoque, separando os produtos por linha, da seguinte forma:

Linha_Prod	Qtd_Estq
1	1200.00
2	700.00

Para exibir esse resultado, deveremos agrupar os produtos por linha de produto e, em seguida, somar a quantidade em estoque de cada um deles.

Observemos o comando a seguir;

```
SELECT LINHA_PROD, SUM(QTD_PROD) AS QTD_ESTQ
FROM PRODUTO
GROUP BY LINHA_PROD
```

No próximo exemplo, suponhamos que seja necessário exibir a quantidade de produtos em estoque, separando-os por tipo e por linha, da seguinte forma:

Tipo_Prod	Linha_Prod	Qtd_Estq
A	1	300
B	1	300
C	1	600
A	2	400
B	2	200
C	2	100

Para conseguirmos exibir esse resultado, deveremos agrupar os produtos por tipo e por linha e, em seguida, somar a quantidade em estoque de cada um deles.

Observemos o comando a seguir:

```
SELECT TIPO_PROD, LINHA_PROD, SUM(QTD_PROD) AS QTD_ESTQ
FROM PRODUTO
GROUP BY LINHA_PROD, TIPO_PROD
```

No próximo exemplo, suponhamos que seja necessário exibir o valor em estoque dos produtos, separando-os por tipo, da seguinte forma:

Tipo_Prod	Val_Estq
A	290000.00
B	385000.00
C	665000.00

Para conseguirmos exibir esse resultado, deveremos agrupar os produtos por tipo de produto e, em seguida, somar o resultado da multiplicação da quantidade de produtos em estoque pelo valor de cada produto.

Observemos o comando seguinte:

```
SELECT TIPO_PROD, SUM(QTD_PROD * VAL_PROD ) AS VAL_ESTQ
FROM PRODUTO
GROUP BY TIPO_PROD
```

Para exibir o valor de produtos em estoque separando-os por linha de produto, como mostra a tabela, deveremos agrupar os produtos por linha e, em seguida, somar o resultado da multiplicação da quantidade de produtos em estoque pelo valor de cada produto.

Linha_Prod	Val_Estq
1	1185000.00
2	155000.00

Observemos:

```
SELECT LINHA_PROD, SUM(QTD_PROD * VAL_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY LINHA_PROD
```

Para exibirmos o valor total em estoque dos produtos, separando-os por tipo e por linha, como mostra o quadro seguinte, deveremos agrupar os produtos por tipo e por linha e, em seguida, somar a multiplicação da quantidade de produtos em estoque pelo valor de cada produto. Vejamos o comando:

```
SELECT TIPO_PROD, LINHA_PROD,
       SUM(QTD_PROD * VAL_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY TIPO_PROD, LINHA_PROD
```

Para exibirmos a quantidade e o valor em estoque dos produtos, separando-os por nome, como mostra o quadro a seguir, será necessário agruparmos os produtos por nome e somarmos a quantidade dos mesmos. Depois, somar a multiplicação da quantidade pelo valor destes mesmos produtos.

Observemos:

Nome_Prod	Qtd_Estq	Val_Estq
Armário	500	220000.00
Cama	600	515000.00
Guarda-Roupa	100	150000.00
Mesa	200	70000.00
Poltrona	100	25000.00
Sofá	400	360000.00

```
SELECT NOME_PROD,
       SUM(QTD_PROD) AS QTD_ESTQ,
       SUM(QTD_PROD * VAL_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY NOME_PROD
```

No exemplo a seguir, para exibirmos o tipo do produto, a linha do produto e o valor em estoque, será necessário agruparmos por tipo, por linha e somarmos a multiplicação da quantidade pelo valor destes mesmos produtos.

Observemos:

Tipo_Prod	Linha_Prod	Val_Estq
A	1	210000.00
B	1	325000.00
C	1	650000.00
A	2	80000.00
B	2	60000.00
C	2	15000.00

```
SELECT TIPO_PROD, LINHA_PROD,
       SUM(QTD_PROD * VAL_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY LINHA_PROD, TIPO_PROD
```

10.1 - GROUP BY/HAVING

A cláusula **HAVING** restringe os dados a serem apresentados. Por exemplo, suponhamos que seja necessário exibirmos a quantidade total em estoque de cada tipo de produto, mas que queiramos apresentar apenas os produtos cuja totalização seja superior a 500 unidades.

Para conseguirmos exibir esse resultado, deveremos agrupar os produtos por tipo de produto, somar a quantidade em estoque de cada um deles e, depois, restringir a totalização com a cláusula. Ao executarmos o comando, os dados que se deseja apresentar ficarão expostos no quadro a seguir:

Tipo_Prod	Qtd_Estq
A	700
C	700

```
SELECT TIPO_PROD, SUM(QTD_PROD) AS QTD_ESTQ
FROM PRODUTO
GROUP BY TIPO_PROD
HAVING SUM(QTD_PROD) > 500
```

A cláusula **HAVING** só poderá ser utilizada juntamente com a cláusula **GROUP BY**.

Observemos a seguir os comandos com a cláusula **HAVING** e os dados que estes apresentam, de acordo com a tabela exemplo **Produto**:

```
SELECT TIPO_PROD, LINHA_PROD, SUM(QTD_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY TIPO_PROD, LINHA_PROD
HAVING SUM(QTD_PROD) > 300
```

Tipo_Prod	Linha_Prod	Qtd_Estq
A	2	400
C	1	600

```
SELECT NOME_PROD, SUM(QTD_PROD) AS QTD_ESTQ,
SUM(QTD_PROD * VAL_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY NOME_PROD
HAVING SUM(QTD_PROD) >= 500
AND SUM(QTD_PROD * VAL_PROD) > 220000.00
```

Nome_Prod	Qtd_Estq	Val_Estq
Cama	600	515000.00

10.2 – GROUP BY / WITH ROLLUP

A cláusula **WITH ROLLUP** é utilizada junto com a cláusula **GROUP BY** para gerar totais e subtotais de acordo com as colunas agrupadas.

Observemos como o resultado a seguir pode ser interessante:

Tipo_Prod	Linha_Prod	Nome_Prod	Qtd_Estq	Val_Estq
A	1	Armário	200	160000.00
A	1	Mesa	100	50000.00
A	Sub-Total Linha 1 Tipo A		300	210000.00
A	2	Armário	300	60000.00
A	2	Mesa	100	20000.00
A	Sub-Total Linha 2 Tipo A		400	80000.00
Sub-Total Tipo A			700	290000.00
B	1	Poltrona	100	25000.00
B	1	Sofá	200	300000.00
B	Sub-Total Linha 1 Tipo B		300	325000.00
B	2	Sofá	200	60000.00
B	Sub-Total Linha 2 Tipo B		200	60000.00
Sub-Total Tipo B			500	385000.00
C	1	Cama	500	500000.00
C	1	Guarda-Roupa	100	150000.00
C	Sub-Total Linha 1 Tipo C		600	650000.00
C	2	Cama	100	15000.00
C	Sub-Total Linha 2 Tipo C		100	15000.00
Sub-Total Tipo C			700	665000.00
Total Geral			1900	1340000.00

As cláusulas **GROUP BY / WITH ROLLUP** podem gerar um resultado bem próximo a este apresentado anteriormente, com a diferença de que as palavras "Sub-Total" e "Total-Geral" não serão apresentadas. No lugar dessas palavras, o MSSQL apresentará a palavra NULL, mas as totalizações obviamente serão as mesmas.

Observemos como o MSSQL apresentará esse resultado:

TIPO_PROD	LINHA_PROD	NOME_PROD	QTD_ESTQ	VAL_ESTQ
A	1	Armário	200	160000.00
A	1	Mesa	100	50000.00
A	1	NULL	300	210000.00
A	2	Armário	300	60000.00
A	2	Mesa	100	20000.00
A	2	NULL	400	80000.00
A	NULL	NULL	700	290000.00
B	1	Poltrona	100	25000.00
B	1	Sofá	200	300000.00
B	1	NULL	300	325000.00
B	2	Sofá	200	60000.00
B	2	NULL	200	60000.00
B	NULL	NULL	500	385000.00
C	1	Cama	500	500000.00
C	1	Guarda-Roupa	100	150000.00
C	1	NULL	600	650000.00
C	2	Cama	100	15000.00
C	2	NULL	100	15000.00
C	NULL	NULL	700	665000.00
NULL	NULL	NULL	1900	1340000.00

O comando para conseguirmos o resultado demonstrado é este:

```
SELECT TIPO_PROD,
       LINHA_PROD,
       NOME_PROD,
       SUM(QTD_PROD) AS QTD_ESTQ,
       SUM(QTD_PROD * VAL_PROD) AS VAL_ESTQ
FROM PRODUTO
GROUP BY TIPO_PROD, LINHA_PROD, NOME_PROD
WITH ROLLUP
```

10.3 – GROUP BY COM JOIN

Para exemplificarmos o uso da cláusula **GROUP BY** com a cláusula **JOIN** no mesmo **SELECT**, vamos utilizar as duas tabelas apresentadas a seguir:

Linha

Linha_Prod	Nome_Linha
1	Primeira Linha
2	Segunda Linha
3	Terceira Linha

TipoProduto

Tipo_Prod	Nome_Tipo
A	Cozinha
B	Sala
C	Quarto

Produto

Cod_Prod	Tipo_Prod	Linha_Prod	Nome_Prod	Qtd_Prod	Val_Prod
1	A	1	Mesa	100	500.00
2	B	1	Sofá	200	1500.00
3	A	2	Mesa	100	200.00
4	A	2	Armário	300	200.00
5	C	1	Cama	500	1000.00
6	B	1	Poltrona	100	250.00
7	B	2	Sofá	200	300.00
8	C	2	Cama	100	150.00
9	A	1	Armário	200	800.00
10	C	1	Guarda_Roupa	100	1500.00

Para exibirmos a quantidade total de produtos, separando-os por linha, e exibindo inclusive o nome da linha, será necessário utilizarmos **JOIN** e **GROUP BY**.

O resultado desejado será este:

Linha_Prod	Nome_Linha	Qtd_Estq
1	Primeira Linha	1200.00
2	Segunda Linha	700.00

Para conseguirmos esse resultado será necessário associarmos as tabelas Produto e Linha com **JOIN**. Para totalizarmos os dados, será necessário utilizarmos a cláusula **GROUP BY**.

Observemos:

```
SELECT PRODUTO.LINHA_PROD, NOME_LINHA,
       SUM(QTD_PROD) AS QTD_ESTQ
FROM PRODUTO
INNER JOIN LINHA
       ON PRODUTO.LINHA_PROD = LINHA.LINHA_PROD
GROUP BY PRODUTO.LINHA_PROD
```

Para exibirmos a quantidade total de produtos, separando-os por linha, exibindo o nome da linha e o nome do tipo dos produtos, será necessário utilizarmos dois **JOINS** e **GROUP BY**.

O resultado desejado será este:

Tipo_Prod	Nome_Tipo	Linha_Prod	Nome_Linha	Qtd_Estq
A	Cozinha	1	Primeira Linha	300
B	Sala	1	Primeira Linha	300
C	Quarto	1	Primeira Linha	600
A	Cozinha	2	Segunda Linha	400
B	Sala	2	Segunda Linha	200
C	Quarto	2	Segunda Linha	100

Para conseguirmos obter o resultado anterior, executamos este comando:

```
SELECT PRODUTO.TIPO_PROD, NOME_TIPO, PRODUTO.LINHA_PROD,
       NOME_LINHA, SUM(QTD_PROD) AS QTD_ESTQ
FROM LINHA
INNER JOIN PRODUTO
      ON LINHA.LINHA_PROD = PRODUTO.LINHA_PROD
INNER JOIN TIPOPROD
      ON PRODUTO.TIPO_PROD = TIPOPROD.TIPO_PROD
GROUP BY TIPOPROD.TIPO_PROD, TIPOPROD.NOME_TIPO,
        LINHA.LINHA_PROD, LINHA.NOME_LINHA
```

Exercícios: (Utilize o database DB_CDS)

- 1 – Exibir quantos pedidos cada cliente fez;
- 2 – Exibir quantos CDs você possui em cada categoria;
- 3 – Exibir quantos CDs você possui de cada gravadora;
- 4 – Exibir quantos pedidos cada funcionário atendeu;
- 5 – Exibir quantos dependentes cada funcionário possui;
- 6 – Exibir quantos pedidos cada cliente fez, mostrando o nome dos clientes;
- 7 – Exibir quando CDs você possui em cada categoria, mostrando os nomes das mesmas;
- 8 – Exibir quantos CDs você possui de cada gravadora, mostrando os nomes das mesmas;
- 9 – Exibir quantos pedidos cada funcionário atendeu, mostrando o nome dos funcionários;
- 10 – Exibir quantos dependentes cada funcionário possui, mostrando os nomes dos funcionários.

11 – Linguagem SQL

A linguagem SQL é dividida em três grupos de comandos:

DCL Data Control Language	DDL Data Definition Language	DML Data Manipulation Language
GRANT	CREATE	SELECT *
DENY	ALTER	INSERT
REVOKE	DROP	UPDATE
		DELETE

* Apesar do SELECT pertencer ao grupo DML, muitos autores o colocam no grupo DQL – Data Query Language, já que o SELECT não manipula dados, somente os consulta.

Os comandos DCL são utilizados para tratar as permissões que os usuários terão dentro de um sistema. Os da DDL são para criação dos objetos do sistema. Já os comandos DML são utilizados no tratamento dos dados de um sistema.

Até aqui, vimos com detalhes os comandos DML e alguns comandos DCL e DDL.

Veremos com detalhes agora, os comandos DDL.

Comandos DDL (Data Definition Language):

Comando CREATE:

Este comando é utilizado para criar banco de dados, tabelas, índices, Triggers entre outros.

Vejamos a criação de banco de dados:

```
CREATE DATABASE NOME_BANCO
```

Este comando cria um novo banco de dados no MSSQL. Para verificar os bancos de dados disponíveis, basta digitar o comando:

```
sp_databases
```

Para excluir um banco de dados, utiliza-se o comando:

```
DROP DATABASE NOME_BANCO
```

Lembrando que após a exclusão do banco de dados, todos os dados serão perdidos.

Comando para criação de tabelas:

```

CREATE TABLE table_name
    ( { < column_definition > | < table_constraint > } [ ,...n ]
    )
< column_definition > ::=
    { column_name data_type }
    [ { DEFAULT constant_expression
      | [ IDENTITY [ ( seed , increment ) ]
      ]
    } ]
    [ ROWGUIDCOL ]
    [ < column_constraint > [ ...n ] ]
< column_constraint > ::=
    [ CONSTRAINT constraint_name ]
    { [ NULL | NOT NULL ]
      | [ PRIMARY KEY | UNIQUE ]
      | REFERENCES ref_table [ ( ref_column ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
    }
< table_constraint > ::=
    [ CONSTRAINT constraint_name ]
    { [ { PRIMARY KEY | UNIQUE }
      { ( column [ ,...n ] ) }
      ]
      | FOREIGN KEY
      ( column [ ,...n ] )
      REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
    }

```

Verifique os parâmetros no help do MSSQL

Para verificar as tabelas disponíveis em um banco de dados, basta digitar o comando:

```

SELECT *
FROM INFORMATION_SCHEMA.TABLES

```

Para verificar informações com mais detalhes de uma tabela, deve-se digitar o comando:

```

SELECT *
FROM INFORMATION_SCHEMA.COLUMNS

```

Para excluir uma tabela, utiliza-se o comando:

```

DROP TABLE NOME_TABELA

```

Lembrando que após a exclusão da tabela, todos os dados da tabela serão perdidos.

Uma tabela, depois de criada, poderá ser modificada, seja com o acréscimo de colunas ou índices.

```
ALTER TABLE table_name
{ [ ALTER COLUMN column_name
  {DROP DEFAULT
  | SET DEFAULT constant_expression
  | IDENTITY [ ( seed , increment ) ]
  }
| ADD
  { < column_definition > | < table_constraint > } [ ,...n ]
| DROP
  { [ CONSTRAINT ] constraint_name
  | COLUMN column }
] }
< column_definition > ::=
  { column_name data_type }
  [ [ DEFAULT constant_expression ]
    | IDENTITY [ ( seed , increment ) ]
  ]
  [ROWGUIDCOL]
  [ < column_constraint > ] [ ...n ] ]
< column_constraint > ::=
  [ NULL | NOT NULL ]
  [ CONSTRAINT constraint_name ]
  {
    | { PRIMARY KEY | UNIQUE }
    | REFERENCES ref_table [ (ref_column) ]
    [ ON DELETE { CASCADE | NO ACTION } ]
    [ ON UPDATE { CASCADE | NO ACTION } ]
  }
< table_constraint > ::=
  [ CONSTRAINT constraint_name ]
  { [ { PRIMARY KEY | UNIQUE }
    { ( column [ ,...n ] ) }
    | FOREIGN KEY
      ( column [ ,...n ] )
      REFERENCES ref_table [ (ref_column [ ,...n ] ) ]
    [ ON DELETE { CASCADE | NO ACTION } ]
    [ ON UPDATE { CASCADE | NO ACTION } ]
  ] }
```

12 - Transações

O que são transações?

No contexto de um sistema e gerenciamento de banco de dados, uma transação é uma seqüência de instruções que dever ser tratadas como uma unidade indivisível, ou seja, ou a ação definida na transação é realizada por completo ou não é realizada.

Este conceito é conhecido como atomicidade. Uma transação é atômica porque não pode ser dividida em partes: é totalmente processada ou totalmente ignorada.

Isto tem implicações particulares se considerarmos o acesso simultâneo por diversos usuários, processos ou threads e também para a recuperação. Cada um destes usuários, programas, processos ou threads podem solicitar acesso ao servidor de banco de dados. Os diversos threads não podem interferir nos outros que estão sendo executados simultaneamente. Se um erro ocorrer, o banco de dados terá que respeitar as transações ao realizar o processo de recuperação. Isto significa retornar o banco de dados ao estado no qual estava antes do erro acontecer ou terminar e anular a transação inteira.

Segue alguns conceitos sobre banco de dados.

Compatibilidade com ACID

Em banco de dados, um termo importante é o acrônimo ACID. ACID significa Atomicidade, Consistência, Isolamento e Durabilidade.

A Atomicidade significa que as transações são atômicas e indivisíveis. Todas as alterações de uma transação serão armazenadas no banco de dados ou nenhuma será armazenada.

A Consistência significa que as operações transformam o banco de dados de um estado válido para outro estado válido. Não há nenhum estágio intermediário onde os dados sejam inconsistentes. O banco de dados também deve desaprovar as operações que violam os limites da consistência. Se estiver armazenando as contas bancárias que se relacionam aos clientes do banco, não deverá ser possível criar uma conta para um cliente que não existe, da mesma forma que não deverá ser possível apagar um cliente da tabela CLIENTE se ainda existirem contas associadas à ele na tabela CONTA.

O Isolamento significa que as transações não se afetam enquanto são executadas. Cada transação deve ser capaz de exibir o mundo como se fosse a única, lendo e alterando as estruturas do banco de dados. Na prática, este não é o caso, mas os locks são utilizados para conseguir esta ilusão.

A Durabilidade significa que depois de uma transação ter sido aceita no banco de dados, seus efeitos são permanentes. Isto seria uma exigência bem simples de satisfazer em um programa pouco complexo, mas em um RDMS complexo, que usa locks, versões diversas para permitir o acesso simultâneo de vários usuários e o cache para melhorar o desempenho, isto é um campo minado. E mais, a durabilidade implica que o sistema recupere o estado anterior do banco de dados em caso de falha. Se uma falha de energia, uma paralisação do disco rígido ou qualquer catástrofe acontecer entre o momento em que um cliente envia uma transação e o momento em que é feito o registro desta transação no disco, o sistema deve ser capaz de gerenciar um backup e um registro, para trazer o banco de dados de volta ao estado anterior à paralisação, juntamente com as transações do processo que foram registradas, mas ainda não executadas ou aceitas.

Dito isto, vejamos como realizar uma transação:

Uma instrução SQL sempre é atômica, ou seja, não é divisível, como o exemplo abaixo:

```
UPDATE CONTA
SET SALDO = SALDO + 500
WHERE NROCONTA = 2
```

No caso acima, houve um depósito na conta.

```
UPDATE CONTA
SET SALDO = SALDO - 500
WHERE NROCONTA = 5
```

No caso acima, houve uma retirada na conta.

Agora, imagine que o cliente vai transferir o dinheiro de uma conta para outra. Se a operação de retirada fosse negada, devido ao saldo do cliente não ser suficiente, o banco de dados ficaria inconsistente, visto que houve um crédito, porém não houve um débito correspondente.

Usando a transação, caso uma operação não seja realizada, a outra é automaticamente cancelada.

Vejamos o exemplo:

```
BEGIN TRANSACTION
UPDATE CONTA
    SET SALDO = SALDO + 500
    WHERE NROCONTA = 2
UPDATE CONTA
    SET SALDO = SALDO - 500
    WHERE NROCONTA = 5
SELECT SALDO
    FROM CONTA
    WHERE NROCONTA = 5
-- A CONSULTA MOSTRARÁ QUE A CONTA 5 FICOU NEGATIVA.
-- A TRANSAÇÃO DEVERÁ SER CANCELADA
ROLLBACK
```

O comando ROLLBACK cancela a transação e irá desfazer qualquer alteração que tenha sido realizada. Uma transação que foi retornada sem sucesso não deixa nenhum rastro nos dados das tabelas.

```
BEGIN TRANSACTION
UPDATE CONTA
    SET SALDO = SALDO + 500
    WHERE NROCONTA = 2
UPDATE CONTA
    SET SALDO = SALDO - 500
    WHERE NROCONTA = 5
SELECT SALDO
    FROM CONTA
    WHERE NROCONTA = 5
-- A CONSULTA MOSTRARÁ QUE A CONTA 5 FICOU POSITIVA.
-- A TRANSAÇÃO ESTÁ CONFIRMADA
COMMIT
```

O comando COMMIT confirma a transação.

Resumindo:

- Uma transação começa em `START TRANSACTION` e termina com `ROLLBACK` ou `COMMIT`. Dentro de um programa, o programador poderá colocar decisões para confirmar ou não a transação.
- Uma transação é considerada Atômica, visto que enquanto não for finalizada com o comando `ROLLBACK` ou `COMMIT`.

13 - TRIGGERS:

Triggers ou gatilhos, são utilizados quando um determinado evento ocorre em uma tabela e o banco de dados automaticamente executa uma instrução previamente cadastrada.

Por exemplo, suponha que sempre que uma tabela tiver uma alteração, o banco de dados gerará um log sobre esta alteração.

```
-- CRIACAO DO BANCO DE DADOS:
CREATE DATABASE EX_TRIGGER
GO
USE EX_TRIGGER
GO
CREATE TABLE CLIENTE
(
    COD_CLI INT IDENTITY PRIMARY KEY,
    NOM_CLI CHAR(30)
)
GO
CREATE TABLE HISTORICOCLIENTE
(
    COD_HIS INT IDENTITY PRIMARY KEY,
    DAT_HIS DATETIME,
    COD_CLI INT,
    NOM_CLI CHAR(30),
    USU_HIS VARCHAR(100)
)
GO
CREATE TRIGGER ALTERA_CLIENTE ON CLIENTE
FOR UPDATE, DELETE
AS
    DECLARE @COD_CLI AS INT
    DECLARE @NOM_CLI AS CHAR(30)
    BEGIN
        SET @COD_CLI = (SELECT COD_CLI FROM INSERTED)
        SET @NOM_CLI = (SELECT NOM_CLI FROM INSERTED)
        INSERT INTO HISTORICOCLIENTE
            (DAT_HIS, COD_CLI, NOM_CLI, USU_HIS) VALUES
            (GETDATE(), @COD_CLI, @NOM_CLI, USER)
    END
```

Podemos criar trigger somente para comandos SQL do grupo DML (INSERT, DELETE E UPDATE).

O MSSQL cria duas tabelas temporárias quando da execução destes comandos, sendo:

A tabela INSERTED

Esta tabela é criada quando da inserção e/ou atualização de dados. São os dados que serão gravados nas tuplas correspondentes. Tem a mesma estrutura da tabela que está tendo tuplas inseridas/alteradas.

A tabela DELETED

Esta tabela é criada quando da exclusão e/ou atualização de dados. São os dados que serão excluídos das tuplas correspondentes. Tem a mesma estrutura da tabela que está tendo tuplas excluídas/alteradas.

Logo após o término do comando, estas tabelas são excluídas, ou seja, seus dados estão disponíveis somente no momento da execução do comando. Lembrando de transações, o comando termina quando seus triggers terminarem. Caso ocorra erro na execução do trigger, o comando é cancelado.

Estrutura do Trigger:

```
CREATE TRIGGER [ schema_name . ] trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
{ sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ]
> }
<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Excluindo um Trigger

Para excluir um trigger, utiliza-se o comando DROP.

```
DROP TRIGGER nome_Trigger
```

O trigger, diferentemente da tabela, não pode ser alterado depois de criado.

14 - PROCEDURES e FUNÇÕES:

Procedures são rotinas que agilizam o acesso ao banco de dados, inclusive criando proteções adicionais ao banco de dados. Diferente de rotinas de linguagem de programação, as procedures do MSSQL podem retornar valores e as funções não podem alterar o banco de dados. As funções são utilizadas para tarefas específicas, tais como retorna a data atual. Não será objeto de nosso estudo, mas caso deseje obter maiores informações, basta verificar na ajuda do MSSQL.

Segue a estrutura básica da Procedure.

```
CREATE PROCEDURE | PROC nome_procedure
    [declaração das variáveis]
    [variável de saída OUTPUT | OUT]
BEGIN
    [comandos sql]
END
```

Exemplo:

```
CREATE DATABASE EX_PROC
GO
USE EX_PROC
GO
CREATE TABLE CLIENTE
(
    COD_CLI INT IDENTITY,
    NOM_CLI CHAR(30)
)
CREATE PROCEDURE SP_GRAVAR_CLIENTE
    @NOME CHAR(30) = NULL
    @RETURN INT OUTPUT
    -- este comando impede o retorno de linhas afetadas
    SET NOCOUNT ON
AS
BEGIN
    INSERT INTO CLIENTE (NOM_CLI) VALUES
        (@NOME)
    SET @RETURN = @@IDENTITY
END
RETURN @RETURN
GO
```

Para excluir uma procedure, utilizamos o comando DROP

```
DROP PROCEDURE nome_procedure
```

15 – BIBLIOGRAFIA

Para confecção desta apostila foi utilizados as seguintes fontes:

- SQL – Guia Prático
Celso Poderoso de Oliveira
- Ajuda On-Line do MS SQL Server
- Apostila do Curso de Banco de Dados Impacta Tecnologia
- Apostilas do site Apostilando.com
- Exemplos do site guiadohardware.com.br
- Fóruns de discussão:
 - Baboo
 - Linha de defesa
 - iMaster