Banco de dados II

Banco de dados II

Douglas Fugita de Oliveira Cezar

Dados Internacionais de Catalogação na Publicação (CIP)

Cezar, Douglas Fujita de Oliveira
C425b Banco de dados II / Douglas Fujita de Oliveira Cezar.
Londrina: Editora e Distribuidora Educacional S.A., 2017. –
56 p.

ISBN 978-85-8482-867-8

1. Banco de dados. I. Título.

CDD 005.7

2017
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Tema 1 | Criação de estruturas e gerenciamento de acesso 5
Tema 2 | Consultas e união de consultas 17
Tema 3 | Alteração, exclusão e o gerenciamento de transações 34
Tema 4 | *Data Warehouse e Data Mining* 49

Convite à leitura

Neste tema você terá o primeiro contato com a linguagem utilizada na programação de banco de dados, a *Structured Query Language* (SQL). Aqui você aprenderá como criar as estruturas básicas de um banco de dados, desde a criação do próprio banco, passando pela criação das tabelas e seus campos, chegando até nas *constraints*, que são restrições que visam garantir a integridade dos dados que serão armazenados nas tabelas.

Este material será importante para você que deseja compreender uma das principais camadas do desenvolvimento de sistemas: a camada de dados, que é a responsável por armazenar e garantir a integridade de toda informação que trafega pelo sistema.

Utilize os exemplos deste caderno de estudos como guia, e crie também sua própria estrutura de banco de dados para que possa avançar nos estudos.

Criação de estruturas e gerenciamento de acesso

POR DENTRO DO TEMA

Uma linguagem de programação de banco de dados pode ser dividida em três grandes grupos: a linguagem de definição de dados (DDL – *Data Definition Language*), a linguagem de controle de dados (DCL – *Data Control Language*) e a linguagem de manipulação de dados (DML – Data *Manipulation Language*).

Através da linguagem de definição de dados é possível que todas as estruturas básicas de um banco de dados sejam criadas, desde uma simples tabela até toda a configuração do arquivo que irá armazenar todo o banco de dados.

Você poderá criar uma base dados através do comando CREATE DATABASE. No Quadro 1.1 você poderá acompanhar a criação da base de dados que abrigará todos os componentes que serão utilizados como exemplo por este caderno e, logo em seguida, a explicação dos comandos.

Quadro 1.1 | Create Database

```
CREATE DATABASE [BD_TADS]
    ON (PRIMARY NAME = 'BD_TADS_
FILE',
        FILENAME = 'Z:\Aula_BD_TADS\',
        SIZE = 5120KB,
        MAXSIZE = UNLIMITED ,
        FILEGROWTH = 1MB )
   LOG ON ( NAME = 'BD_TADS_LOG',
        FILENAME = ' Z:\Aula_BD_TADS\',
```

```
    SIZE = 5120KB,
    MAXSIZE = UNLIMITED ,
    FILEGROWTH = 1MB )
COLLATE Latin1_General_BIN
```

Criar ou definir uma base de dados é a atividade de dar um nome a ela, bem como definir o local em que ela será armazenada e o tamanho dos arquivos que a compõe. Toda base de dados deve ter ao menos um arquivo de dados e um **arquivo de log**. O argumento BD_TADS após o comando CREATE DATABASE define o nome do novo banco de dados.

Em seguida, pode ser observada uma sessão iniciada por ON e o atributo PRIMARY. Esta sessão contém informações sobre o arquivo que armazenará todas as tabelas da base de dados, também chamado de arquivo de dados primário. FILENAME é o atributo no qual deverá ser indicado o local para armazenamento do arquivo de dados primário. Na sequência, temos as propriedades SIZE, MAXSIZE e FILEGROWTH, que estão diretamente relacionados ao tamanho do arquivo. No exemplo do Quadro 1.1, a base está sendo criada com 5120Kb (5MB), possui tamanho máximo ilimitado e, sempre que for necessário aumentar o arquivo, ele crescerá numa taxa de 20%.

A próxima sessão inicia-se com LOG ON, e as propriedades desta sessão são semelhantes à anterior. No entanto, todas as informações são relacionadas à criação do arquivo de log.

Por último, você pode ver o atributo COLLATE, que está relacionado aos caracteres permitidos para esta base de dados, bem como as regras que serão utilizadas para manipulação dos dados. O esquema "Latin1_General_BIN" indica que serão utilizados os conjuntos latinos de caracteres e que a ordem de classificação será binária, além de diferenciar caracteres maiúsculos e minúsculos e acentuados ou não.

Uma vez que o arquivo esteja criado, ele pode ser alterado utilizando o comando ALTER DATABASE. O Quadro 1.2 mostra um exemplo de alteração da base criada.

Quadro 1.2 | ALTER DATABASE

```
ALTER DATABASE [BD_TADS]
  MODIFY FILE ( NAME = 'BD_TADS_log',
   SIZE = 204800KB )
```

O Quadro 1.2 mostra a alteração do tamanho do arquivo de LOG para 200MB. Além do exemplo citado, o comando ALTER DATABASE pode alterar não somente as propriedades indicadas no comando CREATE DATABASE, mas também as opções de **estatísticas, cursores**, recuperação e permissão de leitura e escrita no arquivo como um todo.

A base de dados pode ser excluída, juntamente com seus arquivos de dados e de LOG. Para esta função é utilizado o comando DROP DATABASE, e o exemplo é mostrado no Quadro 1.3.

Quadro 1.3 | DROP DATABASE

```
DROP DATABASE [BD_TADS]
```

Após a criação da base de dados, o próximo componente da estrutura básica de um banco de dados que será criado serão as tabelas. As tabelas são compostas por campos e registros (colunas e linhas, respectivamente) e, neste momento, somente serão definidos os campos e seus tipos.

Para a criação da tabela deve ser usado o comando CREATE TABLE e um exemplo do seu uso pode ser visto no Quadro 1.4.

Quadro 1.4 | CREATE TABLE

```
CREATE TABLE [Produto] (
   [ID_Produto]          int IDENTITY NOT NULL,
   [Nome_Produto]        nvarchar(25) NOT NULL,
   [Descricao_Produto]    ntext NOT NULL,
   [Habilitado_Produto]   binary NULL
   )
```

O Quadro 1.4 indica a criação de uma tabela que será a responsável por armazenar as informações referentes aos produtos que serão cadastrados na base de dados.

Neste exemplo, o argumento Produto do comando CREATE TABLE indica o nome da tabela que será criada. A seguir, entre parênteses são indicados os campos desta tabela, sendo eles: ID_Produto, Nome_Produto, Descricão_Produto e Habilitado_Produto. Cada um deles possui um tipo diferente, devido à natureza dos dados que eles serão capazes de armazenar.

O campo ID_Produto será um campo numérico inteiro e gerado automaticamente, e isto é indicado pelos argumentos int e IDENTITY, respectivamente. Este campo também possui uma indicação de NOT NULL, ou seja, não é permitido que os registros que serão armazenados nesta tabela tenham este campo **NULO**.

Os campos Nome_Produto e Descricao_Produto, embora ambos sejam compostos de caracteres alfanuméricos, possuem tipos de dados diferentes. O tipo nvarchar(n) limita o tamanho do campo ao atributo n indicado na declaração do tipo, que, no exemplo do Quadro 1.4 permite a criação de um campo que terá a possibilidade de armazenar 25 caracteres. Já o tipo ntext é um tipo especial que permite um número bem maior de caracteres (2^30 caracteres).

O campo Habilitado_Produto é do tipo binary, pois seu uso será somente para indicar que o produto em questão poderá ser utilizado ou não. A indicação de NULL indica que este campo poderá conter um valor NULO.

Assim como a base de dados, as tabelas também podem sofrer alterações, e isto será feito através do comando ALTER TABLE.

Quadro 1. 5 | ALTER TABLE ADD

```
ALTER TABLE [Produto]
   ADD [ValorVenda_Produto] float NOT NULL,
   [QuantidadeEstoque_Produto] float NOT NULL,
   [UnidadeMedida_Produto] varchar(3) NOT NULL
```

Os mesmos argumentos utilizados para a criação da tabela devem ser utilizados no comando ALTER TABLE junto ao comando ADD, para que as colunas possam ser incluídas na estrutura criada anteriormente.

Os campos ValorVenda_Produto e QuantidadeEstoque_Produto

foram criados para armazenar campos do tipo float, ou seja, números com ponto flutuante. Todos os campos criados nesta alteração de tabela devem respeitar a regra de não armazenar um valor NULO.

Uma coluna também pode ser removida de uma tabela e, para isso, deve ser utilizado o mesmo comando ALTER TABLE, no entanto, isso deve ser feito com o comando DROP, como mostrado no Quadro 1.6.

Quadro 1.6 | ALTER TABLE DROP

```
ALTER TABLE [dbo].[Produto]
   DROP COLUMN [Habilitado_Produto]
```

O exemplo que será utilizado durante este curso utilizará a estrutura pela query, indicada no Quadro 1.7.

Quadro 1.7 | Estrutura da base de dados BD_TADS

```
  CREATE TABLE [Produto] (
   [ID_Produto]          int IDENTITY NOT NULL,
   [Nome_Produto]        nvarchar(25) NOT NULL,
   [Descricao_Produto]    ntext NOT NULL,
   [Habilitado_Produto]   binary NULL
 [ValorVenda_Produto] float NOT NULL,
   [QuantidadeEstoque_Produto] float NOT NULL,
   [UnidadeMedida_Produto] varchar(3) NOT NULL
   )
CREATE TABLE [Clientes] (
   [ID_Cliente]    int IDENTITY NOT NULL,
   [Nome_Cliente]  nvarchar(25) NOT NULL,
   [Telefone_Cliente]     nvarchar(25) NOT NULL,
   [CPF_Cliente] nvarchar(20) NOT NULL,
   [ID_Municipio]  int NOT NULL,
   [ID_Vendedor]   int NOT NULL
   )
```

```
CREATE TABLE [Municipo] (
   [ID_Municipo]      int IDENTITY NOT NULL,
   [Nome_Municipio]   nvarchar(60) NOT NULL,
   [UF_Municipio]     nvarchar(2) NULL
   )
CREATE TABLE [Vendedor] (
   [ID_Vendedor]      int IDENTITY NOT NULL,
   [Nome_Vendedor]    nvarchar(40) NULL,
   [Comissao_Vendedor] float NULL
   )
CREATE TABLE [NF] (
   [Numero_NF]        nvarchar(9) NOT NULL,
   [Serie_NF]         nvarchar(3) NOT NULL,
   [ID_Cliente]       int NOT NULL,
   [ID_Vendedor]      int NOT NULL,
   [DataEmissao_NF]   datetime NOT NULL,
   [DataSaida_NF]     datetime NOT NULL,
   [ValorTotal_NF]    float NOT NULL
   )
CREATE TABLE [NFItens] (
   [Numero_NF]           nvarchar(9) NOT NULL,
   [Serie_NF]            nvarchar(3) NOT NULL,
  [Item_NFItens]         int NOT NULL,
   [ID_Produto]          int NOT NULL,
  [Qtdade_NFItens]       float NOT NULL,
  [PrecoVenda_NFItens]   float NOT NULL,
   [Desconto_NFItens]    float NULL
   )
```

O próximo passo é garantir a integridade dos dados que serão armazenados na base de dados e, para isso, a linguagem SQL conta com as restrições, ou *constraints*. Elas são utilizadas a fim de garantir a integridade dos dados, mesmo em casos de alterações

ou inserções de dados diretamente na base de dados por usuário que tenham permissão para tal.

As restrições podem ser divididas em três tipos: integridade de domínio, integridade de entidade e integridade referencial.

A integridade de domínio está relacionada aos campos de uma tabela e podem especificar qual é o conjunto de dados válidos para aquele campo. Um exemplo de restrição de integridade de domínio que você já viu neste caderno é a proibição de valor nulo para um determinado campo. As restrições deste tipo são: DEFAULT, CHECK e REFERENCES.

Integridade de entidade está relacionada aos registros, e é utilizada para garantir que cada um dos registros possua um identificador que torne possível sua identificação de forma exclusiva. Pode ser considerada uma restrição de integridade de entidade o uso de chaves-primárias. As restrições deste tipo são: PRIMARY KEY e UNIQUE.

A integridade referencial está correlatada aos relacionamentos entre tabelas e deve garantir que o relacionamento entre tabelas ligadas por chaves-primárias / estrangeiras seja mantido, proibindo a alteração de dados em uma tabela sem que as mudanças sejam refletidas na outra tabela. As restrições deste tipo são: FOREIGN KEY e CHECK.

As constraints podem ser definidas durante a criação da tabela, através do comando CREATE TABLE, ou então por uma alteração de tabela existente, através do comando ALTER TABLE.

A restrição de integridade de domínio DEFAULT deve ser utilizada quando um campo deve ser preenchido com um valor padrão e quando este campo for deixado em branco. Você pode observar um exemplo no Quadro 1.8, no qual, sempre que um produto for cadastro e a quantidade em estoque não for informada, o valor será preenchido com 0 (zero).

Quadro 1.8 | *constraint* DEFAULT

```
ALTER TABLE [Produto]
    ADD CONSTRAINT [DF_QuantidadeEstoque_Produto] DEFAULT 0 FOR [QuantidadeEstoque_Produto]
```

A restrição CHECK faz uma verificação dos dados baseados em uma cláusula condicional. O Quadro 1.9 mostra a inclusão de cláusulas CHECK para verificar se, na tabela NFItens, os campos Qtdade_NFItens e PrecoVenda_NFItens possuem valores maiores que 0 (zero) e se o valor total do item é maior que o valor do campo Desconto_NFItens.

Quadro 1.9 – constraint CHECK

```
ALTER TABLE dbo.NFItens
ADD CONSTRAINT CK_Qtdade_NFItens CHECK (Qtdade_NFItens > 0),
CONSTRAINT CK_PrecoVenda_NFItens CHECK (PrecoVenda_NFItens > 0),
CONSTRAINT CK_Desconto_NFItens CHECK (PrecoVenda_NFItens * Qtdade_NFItens > Desconto_NFItens)
```

A restrição PRIMARY KEY é utilizada para identificar quais campos irão compor a chave-primária, ou seja, o identificador único de cada registro. No Quadro 1.10 você pode acompanhar a criação da chave-primária PK_SerieNumeroNF, no qual os campos Numero_NF e Serie_NF são relacionados como componentes da chave-primária.

Quadro 1. 10 | constraint PRIMARY KEY

```
ALTER TABLE NF
ADD CONSTRAINT PK_SerieNumeroNF PRIMARY KEY (Numero_NF, Serie_NF)
```

Ao definir a chave-primária, deve-se ter em mente que somente uma chave pode ser definida por tabela.

A constraint UNIQUE define um conjunto de campos no qual seu valor deve ser único em toda tabela. Ele é semelhante à chave-primária, e é bastante útil quando outra informação deve ser mantida como única, como o exemplo do Quadro 1.11.

Quadro 1. 11 | constraint UNIQUE

```
ALTER TABLE Clientes
ADD CONSTRAINT PK_ID_Cliente PRIMARY KEY (ID_Cliente),
CONSTRAINT U_CPF_Cliente UNIQUE (CPF_Cliente)
```

As restrições FOREIGN KEY e REFERENCES geralmente são utilizadas em conjunto. A primeira é utilizada para indicar o relacionamento entre duas tabelas indicando o campo que será utilizado como chave-estrangeira. A segunda restrição é utilizada para indicar qual será a chave-primária da outra tabela que irá se relacionar com a chave-estrangeira indicada anteriormente. Um exemplo pode ser visto no Quadro 1.12.

Quadro 1. 12 | constraints FOREIGN KEY e REFERENCES

```
ALTER TABLE Vendedor
ADD CONSTRAINT PK_ID_Vendedor PRIMARY KEY (ID_Vendedor)
ALTER TABLE Clientes
ADD CONSTRAINT FK_ID_Vendedor FOREIGN KEY (ID_Vendedor)
    REFERENCES Vendedor(ID_Vendedor)
```

Agora, a estrutura está pronta, estando definido o banco de dados, as tabelas e também os recursos que permitem que seja garantida a integridade dos dados que estão por vir.

ACOMPANHE NA WEB

Relação de parâmetros de collation do MS SQL Server.

- Este site contém todas as possibilidades de collation que podem ser utilizadas na implementação de um banco de dados. Disponível em: <https://goo.gl/RLLmqj>. Acesso em: 26 mar. 2014.

Tipos de dados (Transact SQL).

- Aqui contém a relação de tipos de dados e o detalhamento de cada tipo que pode ser utilizado durante a criação de uma tabela. Disponível em: <https://goo.gl/ixPn4m>. Acesso em: 26 mar. 2014.

AGORA É A SUA VEZ

Instruções:
Agora, chegou a sua vez de exercitar seu aprendizado.

A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

1. Explique o que é a Linguagem de Definição de Dados (DDL) e qual sua função para um banco de dados.

2. Baseado na query a seguir, identifique quais tipos de integridade são atendidas:

```
ALTER TABLE NF
ADD CONSTRAINT PK_SerieNumeroNF PRIMARY KEY (Numero_NF, Serie_NF)
CONSTRAINT U_SerieNumeroNF UNIQUE (Numero_NF, Serie_NF)
CONSTRAINT CK_ValorTotalNF CHECK (ValorTotalNF > 0)
```

a) Integridade referencial.
b) Integridade de domínio.
c) Integridade de entidade.
d) Nenhuma das respostas anteriores.

3. Marque Verdadeiro ou Falso para as afirmações a seguir:
a) ( ) Uma base de dados pode ser criada somente com um arquivo de LOG.
b) ( ) Após a criação da base de dados, sua estrutura nãopode ser alterada.
c) ( ) O comando CREATE TABLE pode ser utilizado tanto para criar arquivos de dados como para criar tabelas.
d) ( ) O atributo COLLATE pode ser utilizado para indicar o tamanho máximo do banco de dados.

4. A estrutura de banco de dados utilizada durante este caderno deve ser complementada/alterada, de forma que você inclua uma tabela para armazenar grupo de produtos que, por sua vez, deve estar relacionada com a tabela produtos.
A tabela de grupo de produtos deve conter os seguintes campos:
- Código do Grupo de Produtos.
- Descrição do Grupo.
- Percentual máximo de desconto para o grupo.

5. Qual seria o comando para alterar a estrutura da base de dados utilizada, mudando a taxa de crescimento do arquivo primário para 200Mb?

FINALIZANDO

Nesse tema você teve a oportunidade de aprender sobre os principais comandos da linguagem de definição de dados.

Estes comandos são responsáveis pela criação do alicerce que sustentará o banco de dados por toda sua vida útil, por isso, a fase da definição da estrutura é bastante importante.

REFERÊNCIAS

MICROSOFT SQL Server. Nome de agrupamento do Windows (Transact – STL). Disponível em: <http://msdn.microsoft.com/pt-br/library/ms188046%28v=sql.105%29.aspx>. Acesso em: 26 mar. 2014.

MICROSOFT SQL Server. **Tipos de dados** (Transact – STL). Disponível em: <http://msdn.microsoft.com/pt-br/library/ms187752.aspx>. Acesso em: 26 mar. 2014.

SILBERSCHATZ, Abrahan; KORTH, Henry; SUDARSHAN, S. **Sistema de Banco de Dados**. Elsevier, 2012.

GLOSSÁRIO

Arquivo de log: arquivo que, por um determinado período de tempo, armazena todas as transações realizadas em um banco de dados.

Estatística: é uma coleção de dados detalhados sobre o banco de dados, e são utilizadas pelo otimizador do banco para que seja possível propor melhores planos de execução.

Cursores: recurso que possibilita a manipulação de uma linha ou um pequeno bloco de dados.

Nulo: campo que não possui nenhum valor.

Convite à leitura

Neste tema você terá o contato com as estruturas que possibilitarão a recuperação das informações que estão inseridas na base de dados.

A forma como os dados armazenados serão visualizados é muito importante e, por isso, você será introduzido a formas de agrupamento de dados, filtragem de registros por campos específicos e também operações como soma, média, valor máximo e mínimo, contagem, etc.

Dados relacionados que estão em diferentes tabelas também serão relacionados através de comandos específicos, e isto também terá um papel bastante importante na forma de visualização pelo usuário final.

Consultas e união de consultas

POR DENTRO DO TEMA

Uma linguagem de programação de banco de dados pode ser dividida em três grandes grupos: a linguagem de definição de dados (DDL – *Data Definition Language*), a linguagem de controle de dados (DCL – *Data Control Language*) e a linguagem de manipulação de dados (DML – *Data Manipulation Language*).

A DML contém uma série de comandos responsáveis por inserir, excluir, alterar e consultar os dados que estão/serão armazenados no banco de dados.

Para este caderno, será considerado que as tabelas já possuem dados e o Quadro 2.1 mostrará os dados que populam a tabela Município.

Quadro 2. 1 | Dados populados na tabela Município

ID_Municipio	Nome Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP

12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS

Na linguagem de programação SQL, a forma de você exibir os dados de uma ou mais tabelas, assim como mostrados no Quadro 2.1, é através do comando SELECT. Para o resultado acima, deverá ser executado o comando como mostrado no Quadro 2.2.

Quadro 2. 2 | Comando SELECT *

```
SELECT *
FROM [dbo].[Municipio]
```

O "*" indica que todos os campos da tabela indicada na cláusula FROM devem ser considerados na consulta. Outra forma de obter o mesmo resultado é indicar um a um, todos os campos que devem fazer parte da consulta, como mostrado no Quadro 2.3.

Quadro 2.3 | SELECT com campos explícitos

```
SELECT [ID_Municipio], [Nome Municipio], [UF_Municipio]
FROM [dbo].[Municipio]
```

Sendo assim, as consultas podem retornar os dados somente para os campos desejados, sendo possível omitir campos que não são necessários para a obtenção do resultado. Um exemplo seria identificar na tabela Município quais são os estados cujas cidades já foram armazenadas, e o comando para isso é mostrado no Quadro 2.4.

```
SELECT [UF_Município]

    FROM [dbo].[Município]
```

Quadro 2.4 | SELECT com campo específico e resultado

UF_Municipio
SP
RJ
MG
PR
SP
RJ

RJ
BA
BA
BA
SP
MT
MT
RS
RS
RS

No entanto, o resultado desta execução poderia ser melhor para o objetivo inicial, que era de encontrar os estados que estavam armazenados na tabela município e, para isso, pode ser utilizada a instrução DISTINCT. Esta instrução tem o papel de omitir todos os resultados duplicados, e o seu uso e resultado podem ser vistos no Quadro 2.5.

```
SELECT DISTINCT [UF_Municipio]

    FROM [dbo].[Municipio]
```

Quadro 2.5 | Uso do DISTINCT

UF_Município
BA
MG
MT
PR
RJ
RS
SP

Pode ser que em uma consulta seja necessário que alguma condição específica seja atendida como, por exemplo, que sejam listadas todas as cidades que estejam no estado do Rio de Janeiro. Neste caso, a cláusula WHERE deve ser utilizada juntamente com argumentos que representem condições a serem satisfeitas.

As condições devem utilizar **operadores de comparação**, tais como

igual a (=), maior que (>), menor que (<) e diferente de (<>). Há também operadores especiais como o BETWEEN, no qual pode ser especificado o intervalo de valores, o LIKE, no qual podem ser especificadas partes de uma palavra e o IN, que utiliza uma lista de valores para a comparação. Várias condições podem ser consideradas em uma mesma cláusula WHERE utilizando os operadores lógicos E (AND), OU (OR) ou NÃO (NOT).

Sendo assim, para filtrar as cidades armazenadas na tabela Municipios que pertencem ao estado do Rio de Janeiro, o comando seria mostrado no Quadro 2.6.

```
SELECT [Nome_Municipio], [UF_Municipio]
    FROM [dbo].[Municipio]
    WHERE UF_Municipio = 'RJ'
```

Quadro 2. 6 | Uso da cláusula WHERE

Nome_Município	UF_Município
Rio de Janeiro	RJ
Angra dos Reis	RJ
Niterói	RJ

Embora todas as consultas mostradas até o momento tivessem como base uma única tabela, as consultas com o comando SELECT podem trazer resultados cruzando informações entre duas ou mais tabelas diferentes. Um caso para demonstrar esta necessidade é o apresentado no Quadro 2.7, no qual uma consulta foi realizada para exibir o nome dos clientes armazenados e o município ao qual eles pertencem.

```
SELECT      ID_Cliente,
            Nome_Cliente,
            ID_Municipio
FROM        [dbo].[Clientes]
```

Quadro 2.7 | SELECT separados mostrando o conteúdo de duas tabelas relacionadas

ID_Cliente	Nome_Cliente	ID_ Municipio
1	CLIENTE A	3
2	CLIENTE B	2
3	CLIENTE C	2
4	CLIENTE D	1
5	CLIENTE E	4
6	CLIENTE F	5

SELECT * FROM Municipio

ID_Municipio	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS

Com estas tabelas podemos dizer que o "CLIENTE A" está no município de "Belo Horizonte" e que o "CLIENTE B", por sua vez, no "Rio de Janeiro". No entanto, seria muito mais **amigável** se estas informações estivessem disponíveis em uma mesma consulta e, para isso, podemos utilizar a chave "ID_Municipio" que está presente nas duas tabelas. Para isso, precisamos listar na cláusula FOR as duas tabelas envolvidas nesta

consulta e o relacionamento será forçado através da cláusula WHERE, como mostrado no Quadro 2.8.

```
SELECT    ID_Cliente,
          Nome_Cliente,
          Nome_Municipio,
          UF_Municipio
FROM      Clientes c, Municipio m
WHERE     c.ID_Municipio = m.ID_Municipio
```

Quadro 2. 8 | SELECT relacionando duas tabelas

ID_Cliente	Nome_Cliente	Nome Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
1	CLIENTE A	Belo Horizonte	MG
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP

Qualquer um dos campos envolvendo estas duas tabelas são passíveis de serem utilizados como filtros através da cláusula WHERE, por exemplo, caso seja necessário descobrir quais clientes estão no estado de São Paulo, como pode ser visto no Quadro 2.9.

```
SELECT    ID_Cliente,Nome_Cliente,Nome_Municipio,UF_Municipio
FROM      Clientes c, Municipio m
WHERE     c.ID_Municipio = m.ID_Municipio
AND       UF_Municipio = 'SP'
```

Quadro 2. 9 | SELECT utilizando campo da tabela relacionada como filtro

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP

6	CLIENTE F	Campinas	SP

Uma outra opção para o relacionamento entre tabelas é utilizar a cláusula JOIN, que explicita uma junção entre tabelas. Os JOINs podem ser divididos em dois tipos: OUTER JOIN e INNER JOIN.

A opção pelo OUTER JOIN deve ser feita sempre que todos os resultados de uma das tabelas devam ser mostrados, mesmo que alguns dos dados não tenham nenhum relacionamento com a outra tabela.

Para relacionar as tabelas com o OUTER JOIN, é necessário saber qual das tabelas terão todos os resultados preservados e isto deverá ser indicado através das três formas de OUTER JOIN: LEFT OUTER JOIN, RIGHT OUTER JOIN e FULL OUTER JOIN.

O LEFT OUTER JOIN preserva todos os resultados da tabela à esquerda do comando e o RIGHT OUTER JOIN, por sua vez, preserva os resultados da tabela à direita. Já o FULL OUTER JOIN preserva o resultado das duas tabelas. Um exemplo da execução do OUTER JOIN pode ser visto no Quadro 2.10.

```
SELECT    ID_Cliente,
          Nome_Cliente,
          Nome_Municipio,
          UF_Municipio
FROM      Clientes c LEFT OUTER JOIN Municipio m
ON        ( c.ID_Municipio = m.ID_Municipio)
```

Quadro 2.10 | LEFT e RIGHT OUTER JOIN

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_ Municipio
1	CLIENTE A	Belo Horizonte	MG
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
4	CLIENTE D	Sao Paulo	SP

5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP

```
SELECT    ID_Cliente,
          Nome_Cliente,
          Nome_Municipio,
          UF_Municipio
FROM      Clientes c RIGHT OUTER JOIN Municipio m
ON        ( c.ID_Municipio = m.ID_Municipio)
```

Quadro 2.11 | LEFT e RIGHT OUTER JOIN

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
1	CLIENTE A	Belo Horizonte	MG
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP
(null)	(null)	Angra dos Reis	RJ
(null)	(null)	Niteroi	RJ
(null)	(null)	Salvador	BA
(null)	(null)	Feira de Santana	BA
(null)	(null)	Porto Seguro	BA
(null)	(null)	Guarulhos	SP
(null)	(null)	Cuiaba	MT
(null)	(null)	Barra do Garça	MT
(null)	(null)	Porto Alegre	RS
(null)	(null)	Uruguaiana	RS
(null)	(null)	Caxias do Sul	RS

No Quadro 2.11, pode ser observado que o LEFT OUTER JOIN preservou todos os resultados da tabela da esquerda (Clientes) e os relacionando com as tabelas da direita (Município). Já o RIGHT OUTER

JOIN preservou todos os resultados da tabela da direita (Município), e os relacionou com os da tabela da esquerda (Clientes). Observe também que, mesmo que não haja relacionamento com a tabela, todos os registros são exibidos, como é o caso dos registros que estão com valor NULL (**nulo**).

Diferentemente do OUTER JOIN, o INNER JOIN somente exibe os registros no qual ambas as tabelas estão relacionadas, como pode ser visto no Quadro 2.12.

SELECT ID_Cliente,Nome_Cliente,Nome_Municipio,UF_Municipio

FROM Municipio m **INNER JOIN** Clientes c

ON (c.ID_Municipio = m.ID_Municipio)

Quadro 2.12 | INNER JOIN

ID_Cliente	Nome_Cliente	Nome_Municipio	UF_Municipio
4	CLIENTE D	Sao Paulo	SP
2	CLIENTE B	Rio de Janeiro	RJ
3	CLIENTE C	Rio de Janeiro	RJ
1	CLIENTE A	Belo Horizonte	MG
5	CLIENTE E	Curitiba	PR
6	CLIENTE F	Campinas	SP

O resultado desta execução é idêntico ao LEFT JOIN, mas não passa de coincidência. O LEFT JOIN foi auxiliado pela CONSTRAINT que obriga o preenchimento do campo ID_Municipio e, por isso, não é possível a criação de um registro na tabela Clientes que não tenha relacionamento com a tabela Municipio.

A utilização de JOINs, ou então do relacionamento forçado através das cláusulas FROM e WHERE, como você pôde acompanhar anteriormente, facilitou a visualização das informações, reunindo dados de várias tabelas em um único conjunto de dados.

Os resultados de uma consulta podem sofrer agregações, como por exemplo, a soma dos valores das vendas para cada cliente ou o valor médio de venda de cada produto. A estrutura do comando SELECT

permite que sejam utilizadas funções de agregação como soma (SUM), contagem (COUNT) e Média (AVG).

Para exemplificar as funções de agregação, serão introduzidas as tabelas do Quadro 2.13, juntamente com o conteúdo das mesmas.

```
SELECT * FROM [dbo].[NF]

GO
```

Quadro 2.13 | Tabelas Produto, Nota Fiscal de Venda e Itens da Nota Fiscal

Numero_ NF	Serie_ NF	ID_ Cliente	ID_ Vendedor	DataEmissao_ NF	DataSaida_NF
5	1	1	1	01/03/2014	02/03/2014
7	1	3	2	01/03/2014	06/03/2014
8	1	1	1	02/03/2014	02/03/2014
9	1	4	1	02/03/2014	04/03/2014
10	1	6	4	04/03/2014	06/03/2014
12	1	2	2	08/03/2014	08/03/2014

```
SELECT * FROM [dbo].[NFItens]

GO
```

Número_ NF	Série_NF	Item_ NFItens	ID_ Produto	Qtdade_ NFItens	PrecoVenda_ NFItens
5	1	1	1	20	2
5	1	2	2	20	4
7	1	1	3	10	3,5
7	1	2	1	20	1,5
7	1	3	5	2	5
8	1	1	4	1	12
9	1	1	6	2	20,5
10	1	1	6	6	16,5
12	1	1	1	10	1,5

```
SELECT * FROM [dbo].[Produto]

GO
```

ID_Produto	Nome_ Produto	ValorVenda_ Produto	UnidadeMedida_ Produto
1	Lapis	1,49	UN
2	Caneta azul	3,5	UN
3	Caderno 100fl	15	UN
4	Grampo	12,9	CX
5	Borracha	4,2	UN
6	Folha Sulfite	18,9	PCT

Você pode observar que as tabelas do Quadro 2.13 fazem referência a várias outras tabelas. Por exemplo, a tabela NF possui campos de chave estrangeira os quais referenciam as tabelas que armazenam dados de clientes e de vendedores. Para que faça sentido os dados apresentados pela função de agregação, as tabelas deverão ser relacionadas por JOINs, como pode ser visto no Quadro 2.14.

```
SELECT    n.Numero_NF as [Numero NF],
          n.Serie_NF as [Serie NF],
          c.Nome_Cliente as Cliente,
          v.Nome_Vendedor as Vendedor,
          n.DataEmissao_NF as [Data de Emissao],
          p.Nome_Produto as [Produto],
          ni.Qtdade_NFItens as [Quantidade],
          ni.PrecoVenda_NFItens as [Preco de Venda]
FROM NF n
INNER JOIN NFItens ni
  ON      n.Numero_NF = ni.Numero_NF AND n.Serie_NF = ni.Serie_NF
INNER JOIN Produto p
  ON      ni.ID_Produto = p.ID_Produto
```

INNER JOIN Clientes c

ON n.ID_Cliente = c.ID_Cliente

INNER JOIN Vendedor v

ON n.ID_Vendedor = v.ID_Vendedor

ORDER BY n.Serie_NF, n.Numero_NF

Quadro 2.14 | Resultado do JOIN das tabelas da figura 2.13 e seus relacionamentos

nº NF	Série NF	Cliente	Vendedor	Data de Emissão	Produto	Qtd.	Preço de Venda
5	1	CLIENTE A	HUGO	01/03/2014	Lapis	20	2
5	1	CLIENTE A	HUGO	01/03/2014	Caneta azul	20	4
7	1	CLIENTE C	LUIZ	01/03/2014	Caderno 100fl	10	3,5
7	1	CLIENTE C	LUIZ	01/03/2014	Lapis	20	1,5
7	1	CLIENTE C	LUIZ	01/03/2014	Borracha	2	5
8	1	CLIENTE A	HUGO	02/03/2014	Grampo	1	12
9	1	CLIENTE D	HUGO	02/03/2014	Folha Sulfite	2	20,5
10	1	CLIENTE F	DONALD	04/03/2014	Folha Sulfite	6	16,5
12	1	CLIENTE B	LUIZ	08/03/2014	Lapis	10	1,5

Com este JOIN apresentado no Quadro 2.14, será feito uma soma do valor das notas fiscais emitidas através da função de agregação SUM e os valores agregados serão agrupados por cliente, como exibido no Quadro 2.15.

SELECT c.Nome_Cliente as Cliente,

```
    SUM     (ni.PrecoVenda_NFItens * ni.Qtdade_NFItens) as
[Preco de Venda]
    FROM    NF n
    INNER JOIN NFItens ni
     ON     n.Numero_NF = ni.Numero_NF AND n.Serie_NF =
ni.Serie_NF
    INNER JOIN Produto p
     ON     ni.ID_Produto = p.ID_Produto
    INNER JOIN Clientes c
     ON     n.ID_Cliente = c.ID_Cliente
   GROUP BY c.Nome_Cliente
   ORDER BY c.Nome_Cliente
```

Quadro 2.15 | Resultado de operação SUM

Cliente	Preco de Venda
CLIENTE A	132
CLIENTE B	15
CLIENTE C	75
CLIENTE D	41
CLIENTE F	99

Ao utilizar uma função de agregação, note que somente os campos principais foram deixados no SELECT. Além disto, você também deve se atentar ao fato de que todos os campos pelo qual o resultado será agrupado deverá ser relacionado na cláusula GROUP BY, como exemplo do Quadro 2.15.

Em um único SELECT pode haver quantas funções de agregação quanto necessárias, bem como agrupar por vários níveis diferentes.

ACOMPANHE NA WEB

Relação de funções de agregação do MS SQL Server

- Relação de funções de agregação do MS SQL Server. Neste site você pode verificar todas as funções de agregação, bem como exemplos de uso e sua sintaxe.

Disponível em: <http://technet.microsoft.com/pt-br/library/ms173454.aspx>. Acesso em: 1 abr. 2014.

AGORA É A SUA VEZ

Instruções:
Agora, chegou a sua vez de exercitar seu aprendizado. A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

1. Explique a importância de exibir as informações de forma amigável ao extrair os dados de um banco de dados.

2. Marque a seguir as funções de agregação:
a) INNER JOIN
b) SUM
c) ALTER
d) LEFT OUTER JOIN
e) AVG
f) COUNT

3. Para preservar todos os dados da primeira tabela, e exibir os dados da segunda tabela que estiverem relacionado com a primeira, deve ser utilizada a seguinte cláusula de agregação:
a) INNER JOIN
b) LEFT OUTER JOIN
c) RIGHT OUTER JOIN
d) FULL OUTER JOIN
e) Nenhuma das respostas anteriores.

4. Utilizando como base a estrutura de dados mostrada neste caderno, crie uma consulta na qual seja possível obter o valor médio de venda de cada produto.

5. Utilizando como base a estrutura de dados mostrada neste caderno, crie uma consulta na qual seja possível identificar quais itens foram comprados por cada cliente e qual a quantidade de compras daquele item.

FINALIZANDO

Neste caderno você teve a oportunidade de aprender sobre o comando SELECT, responsável por recuperar as informações armazenadas nas tabelas do banco de dados.

Foram apresentados conceitos introdutórios que devem ser expandidos através de leituras complementares, pois seu conhecimento é de vital importância para a utilização eficaz de um banco de dados.

REFERÊNCIAS

MICROSOFT SQL Server. **Funções de Agregação** (Transact- SQL). Disponível em: <http://technet.microsoft.com/pt-br/library/ms173454.aspx>. Acesso em: 1 jun. 2014

SILBERSCHATZ, Abrahan; KORTH, Henry; SUDARSHAN, S. **Sistema de Banco de Dados.** Elsevier, 2012.

GLOSSÁRIO

Operadores de comparação: elementos que comparam dois valores e retornam como resultado um valor TRUE (verdadeiro) ou FALSE (falso).

Amigável: que seja feito de forma que o uso da informação seja facilitado.

Nulo: campo que não possui nenhum valor.

Convite à leitura

Neste tema você terá o contato com os comandos responsáveis pela manipulação dos dados do banco de dados, seja através da inclusão, exclusão ou alteração dos registros existentes.

Um conceito que será revisto neste caderno será o da atomicidade, uma das características básicas de um banco de dados. Este conceito define que determinadas operações devem ser executadas por completo e, caso a operação seja interrompida no meio do caminho, os dados que já foram alterados devem voltar ao que eram antes do início da execução.

Será abordado também o conceito de permissões de acessos. Você poderá entender como deve ser feita a concessão de permissão ou então a negação da permissão para um usuário em específico.

Com este conjunto de comandos e conceitos, você estará apto a programar os mais diversos bancos de dados.

Alteração, exclusão e o gerenciamento de transações

POR DENTRO DO TEMA

Uma linguagem de programação de banco de dados pode ser dividida em três grandes grupos: a linguagem de definição de dados (DDL – *Data Definition Language*), a linguagem de controle de dados (DCL – *Data Control Language*) e a linguagem de manipulação de dados (DML – *Data Manipulation Language*).

A DML contém uma série de comandos responsáveis por inserir, excluir, alterar e consultar os dados que estão/serão armazenados no banco de dados e, neste caderno, daremos continuidade aos comandos de manipulação de dados.

Para este caderno, continuaremos utilizando como base as tabelas exibidas anteriormente nos outros temas. O Quadro 3.1 mostrará os dados que populam a tabela Município.

Quadro 3.1 | Dados populados na tabela Município

ID_Municipo	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT

13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS

Para as consultas realizadas anteriormente, foi considerada esta tabela já preenchida com dados. No entanto, existe um comando para que os dados sejam inseridos em uma tabela e este comando é o INSERT.

Para inserir um dado na tabela Municipio, será utilizado o comando mostrado no Quadro 3.2, bem como a tabela após a inclusão.

```
INSERT INTO [dbo].[Municipio]([Nome_Municipio], [UF_Municipio])

    VALUES('Palmas', 'RO')

//Erro proposital, para exemplo futuro
```

Quadro 3.2 | Comando INSERT

ID_Municipo	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP
2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS
17	Palmas	RO

O comando INSERT também pode popular uma tabela baseado em um comando de consulta SELECT. Será criada uma estrutura

para armazenar as siglas dos estados e ela será populada com os estados cujos clientes já receberam pelo menos uma nota fiscal, como pode ser visto no Quadro 3.3. O resultado da operação também será mostrado no quadro.

```
CREATE TABLE [dbo].[UF_Com_NF] (
  [id_UF] int IDENTITY,
  [UF]    varchar(2) NULL
  )
INSERT INTO UF_com_NF
  SELECT DISTINCT UF_Municipio FROM Municipio m
    INNER JOIN Clientes c
      ON c.ID_Municipio = m.ID_Municipo
    INNER JOIN NF n
      ON n.ID_Cliente = c.ID_Cliente
```

Quadro 3.3 | INSERT baseado em SELECT

id_UF	UF
9	MG
10	RJ
11	SP

A informação criada com o INSERT do Quadro 3.2, como identificado no comentário do comando, foi gerada com erro. Para corrigir o erro, será utilizado o comando UPDATE, conforme mostrado no Quadro 3.4.

```
UPDATE Municipio SET UF_Municipio = 'TO'
WHERE Nome_Municipio = 'Palmas'
```

Quadro 3.4 | Comando UPDATE

ID_Municipo	Nome_Municipio	UF_Municipio
1	Sao Paulo	SP

2	Rio de Janeiro	RJ
3	Belo Horizonte	MG
4	Curitiba	PR
5	Campinas	SP
6	Angra dos Reis	RJ
7	Niteroi	RJ
8	Salvador	BA
9	Feira de Santana	BA
10	Porto Seguro	BA
11	Guarulhos	SP
12	Cuiaba	MT
13	Barra do Garça	MT
14	Porto Alegre	RS
15	Uruguaiana	RS
16	Caxias do Sul	RS
17	Palmas	TO

O comando UPDATE irá atualizar todos os registros que forem satisfeitos pela cláusula WHERE. Caso não haja a cláusula WHERE no comando, todos os registros da tabela serão afetados.

Através do comando UPDATE pode ser feita uma atualização no preço de venda dos produtos, utilizando fórmulas para definir o valor a ser atualizado, como pode ser visto no Quadro 3.5.

Quadro 3.5 | UPDATE com fórmula

Consulta antes do UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto
1	Lapis	1,49
2	Caneta azul	3,5
3	Caderno 100fl	15
4	Grampo	12,9
5	Borracha	4,2
6	Folha Sulfite	18,9

```
UPDATE Produto
SET ValorVenda_Produto = ValorVenda_Produto * 1.1
```

Consulta após UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto
1	Lapis	1,64
2	Caneta azul	3,85
3	Caderno 100fl	16,5
4	Grampo	14,19
5	Borracha	4,62
6	Folha Sulfite	20,79

A estrutura CASE também pode ser utilizada para que a atualização possa ser variável. O Quadro 3.6 mostra a diminuição de 25% do valor de venda caso seja comercializado em caixas (CX) ou a diminuição de 5% caso seja comercializado em pacotes (PCT). Para as demais, os preços serão aumentados em R$ 0,10.

Quadro 3.6 | UPDATE com estrutura CASE

Consulta antes do UPDATE

ID_Produto	Nome_Produto	ValorVenda_Produto	UnidadeMedida_Produto
1	Lapis	1,64	UN
2	Caneta azul	3,85	UN
3	Caderno 100fl	16,5	UN
4	Grampo	14,19	CX
5	Borracha	4,62	UN
6	Folha Sulfite	20,79	PCT

```
UPDATE Produto
    SET ValorVenda_Produto = CASE
        WHEN UnidadeMedida_Produto = 'CX'
            THEN ROUND(ValorVenda_Produto*0.75,2)
        WHEN UnidadeMedida_Produto = 'PCT'
```

THEN ROUND(ValorVenda_Produto***0.95,2**)

ELSE ValorVenda_Produto + **0.1**

END

Consulta após UPDATE

ID_ Produto	Nome_Produto	ValorVenda_ Produto	UnidadeMedida_ Produto
1	Lapis	1,74	UN
2	Caneta azul	3,95	UN
3	Caderno 100fl	16,6	UN
4	Grampo	10,64	CX
5	Borracha	4,72	UN
6	Folha Sulfite	19,75	PCT

Outro comando utilizado para a manipulação de dados é o DELETE, responsável pela exclusão de informações de uma tabela. Assim como o UPDATE, ele deve sempre ser utilizado com uma cláusula WHERE para definir quais registros serão afetados. Caso contrário, todos os registros de uma tabela serão apagados. O exemplo pode ser visto no Quadro 3.7.

Quadro 3. 7 | Comando DELETE

Consulta antes do DELETE

id_UF	UF
9	MG
10	RJ
11	SP

DELETE FROM UF_com_NF

Consulta após DELETE

id_UF	UF

É importante reparar que o comando DELETE somente apaga os dados, mas a estrutura da tabela é mantida. Caso queira afetar

também a estrutura da tabela, deve ser utilizado o comando DROP TABLE, já visto anteriormente.

Um das características de um banco de dados é a atomicidade. Este princípio indica que uma operação deve ser realizada por completo e, caso seja interrompida durante a execução, as alterações realizadas anteriormente devem ser descartadas.

O uso do controle de transação está **implícito** nos comandos INSERT, UPDATE ou DELETE quando executados sozinhos, e os registros são alterados somente se o comando for executado com sucesso.

No entanto, existem comandos que possibilitam o controle de transação de forma explícita. São eles: BEGIN TRANSACTION, COMIT TRANSACTION e ROLLBACK TRANSACTION.

O exemplo clássico para este tipo de operação é a transferência eletrônica de dinheiro, na qual é realizada uma operação de saque e outra de depósito, e as duas devem ser realizadas atomicamente, ou seja, uma não pode ser realizada caso a outra não seja. Para exemplificar este contexto, será utilizada a tabela mostrada no Quadro 3.8.

Quadro 3.8 | Tabela Conta Bancária e UPDATE da transferência

Banco	agencia	conta	saldo
5	6658	118779-5	R$ 25.441,00
633	2115	5496085-9	R$ 78.600,00

```
UPDATE ContaBancaria
   SET saldo = saldo - 900
   WHERE banco = '005'
      AND agencia = '6658'
      AND conta = '118779-5'
UPDATE ContaBancaria
   SET saldo = saldo + 900
```

```
WHERE banco = '633'
  AND agencia = '2115'
  AND conta = '5496085-9'
```

No exemplo anterior, caso haja alguma falha após a execução do primeiro UPDATE, a primeira conta será diminuída de R$ 900,00, e este valor não será acrescido na outra. O inverso também é valido, caso somente a segunda atualização seja efetuada.

Para evitar este tipo de falha, o comando de transação é utilizado conforme mostra o Quadro 3.9

Quadro 3.9 | BEGIN TRANSACTION e COMMIT TRANSACTION

```
BEGIN TRANSACTION Trans01
  UPDATE ContaBancaria
    SET saldo = saldo - 900
    WHERE banco = '005'
      AND agencia = '6658'
      AND conta = '118779-5'

  UPDATE ContaBancaria
    SET saldo = saldo + 900
    WHERE banco = '633'
      AND agencia = '2115'
      AND conta = '5496085-9'
COMMIT TRANSACTION Trans01
```

As alterações solicitadas pelos comandos executados abaixo do BEGIN TRANSACTION são realizadas somente após o comando COMMIT TRANSACTION. Caso algum erro ocorra, o comando de COMMIT não é realizado.

O Quadro 3.10 mostra como o banco de dados reage ao ROLLBACK.

Quadro 3.10 | Resultado de um ROLLBACK

```
BEGIN TRANSACTION Trans02

  UPDATE ContaBancaria

  SET saldo = 28000
```

```
    WHERE agencia = '6658';

SELECT * from ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 28.000,00
633	2115	5496085-9	R$ 78.600,00

```
ROLLBACK TRANSACTION Trans02;

SELECT * FROM ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 25.441,00
633	2115	5496085-9	R$ 78.600,00

Embora, após o comando UPDATE, o resultado do comando SELECT mostrasse um saldo atualizado, somente um comando de COMMIT oficializaria este dado na **base de dados**. Após a execução do ROLLBACK, o valor anterior é recuperado e a atualização é descartada. A mesma transação pode ser vista, agora com o comando de COMMIT, no Quadro 3.11.

Quadro 3.11 | Resultado de um COMMIT.

```
BEGIN TRANSACTION Trans03

    UPDATE ContaBancaria

    SET saldo = 28000

    WHERE agencia = '6658';


SELECT * from ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 28.000,00
633	2115	5496085-9	R$ 78.600,00

```
COMMIT TRANSACTION Trans03;
```

```
SELECT * FROM ContaBancaria;
```

banco	agencia	conta	saldo
5	6658	118779-5	R$ 28.000,00
633	2115	5496085-9	R$ 78.600,00

Para automatizar a execução de COMMIT e ROLLBACK, dependendo da execução dos comandos da query, um recurso que pode ser utilizado é a variável global @@ERROR, que mantém seu valor em zero (0) até que um erro aconteça. Com isso, você pode ver no Quadro 3.12 uma forma de disparar os comandos de COMMIT ou ROLLBACK automaticamente.

Quadro 3.12 | Automatização de COMMIT e ROLLBACK.

```
BEGIN TRANSACTION Trans01
    UPDATE ContaBancaria
        SET saldo = saldo - 900
        WHERE banco = '005'
            AND agencia = '6658'
            AND conta = '118779-5'
    UPDATE ContaBancaria
        SET saldo = saldo + 900
        WHERE banco = '633'
            AND agencia = '2115'
            AND conta = '5496085-9'
IF @@ERROR <> 0
    ROLLBACK TRANSACTION Trans01
ELSE
    COMMIT TRANSACTION Trans01
```

Para que os comandos vistos neste caderno possam ser executados, assim como SELECT do caderno anterior, os comandos da DCL, linguagem de controle de dados, devem ser utilizados. Estes comandos servem para dar ou negar o privilégio de acesso aos objetos da base de dados.

A autorização deve ser dada por usuário e, para isso, serão utilizados os comandos CREATE LOGIN e CREATE USER, conforme mostra o Quadro 3.13.

Quadro 3.13 | CREATE LOGIN e CREATE USER

```
CREATE LOGIN novo_usuario WITH PASSWORD = 'nova_senha'

USE BD_TADS

CREATE USER novo_usuario FOR LOGIN novo_usuario
```

A permissão é concedida através do comando GRANT e negada pelo comando DENY. Estes comandos devem ser executados levando em consideração o que será permitido/negado e a quem. As opções mais comuns de permissão/negação são os comandos SELECT, INSERT, UPDATE e DELETE. O uso da permissão GRANT pode ser visto no Quadro 3.14.

Quadro 3.14 | GRANT – concessão de permissão

```
GRANT SELECT
  on Clientes, Municipio
  TO novo_usuario

GRANT SELECT, UPDATE
  ON NF,NFItens
  TO novo_usuario

DENY INSERT, DELETE
  ON NF,NFItens, Clientes, Municipio
  TO novo_usuario
```

No exemplo anterior, o novo_usuário recebeu permissão de visualizar o conteúdo das tabelas Clientes e Municipio, bem como visualizar e alterar as tabelas NF e NFItens. Ao mesmo tempo, está sendo negada permissão de executar os comandos INSERT e DELETE para as tabelas citadas anteriormente.

O comando DENY anula qualquer permissão anterior mesmo que haja um comando GRANT dando permissão ao usuário executar alguma operação em determinada tabela.

Os comandos de permissão podem ser especificados para que somente campos específicos sejam afetados pela permissão. Um exemplo pode ser visto no Quadro 3.15.

Quadro 3.15 | GRANT com campo específico

```
GRANT UPDATE(Desconto_NFItens)
  ON NFItens
  TO novo_usuario
```

Este argumento de utilizar com a permissão UPDATE indica que somente o campo Desconto_NFItens pode ser alterado pelo usuário novo_usuario. Esta mesma estratégia pode ser utilizada pela permissão INSERT.

Uma permissão pode ser removida através do comando REVOKE e sua sintaxe é semelhante à concessão/negação de permissão. Um exemplo é mostrado Quadro 3.16.

Quadro 3.16 | REVOKE

```
REVOKE SELECT
  ON Clientes
  FROM novo_usuario
```

Diferentemente do DENY, o REVOKE não nega uma permissão. Este comando somente retira a permissão concedida/negada, deixando assim as permissões como eram anteriormente à operação de GRANT ou DENY. Caso a tabela Clientes tenha permissão pública negada como opção padrão, o usuário novo_usuario se enquadrará na mesma regra, visto que sua permissão foi retirada.

ACOMPANHE NA WEB

Instruções de transação (Transact-SQL)

- Instruções detalhadas sobre transação no MS SQL Server. Este site contém, detalhadamente, todas as possibilidades de utilização do recurso de transações. Vale a pena aprofundar-se neste assunto para que seu uso seja efetivo.

Disponível em: <http://msdn.microsoft.com/pt-br/library/ms174377.aspx>. Acesso em: 15 abr. 2014.

AGORA É A SUA VEZ

Instruções:
Agora, chegou a sua vez de exercitar seu aprendizado. A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

1. Defina o conceito de atomicidade. Dê um exemplo que ilustre este conceito.

2. Os comandos executados após o BEGIN TRANSACTION são confirmados somente após um comando:
a) END TRANSACTION
b) ROLLBACK TRANSACTION
c) CASE TRANSACTION
d) COMMIT TRANSACTION
e) UPDATE

3. O campo saldo possui o valor inicial de R$ 5.000,00, e as operações a seguir serão realizadas após a declaração de início de uma transação. O campo saldo é alterado, sendo acrescido de R$ 800 e, após, é realizada uma operação de ROLLBACK. Qual o valor no campo saldo antes e depois do ROLLBACK?
a) R$ 5.000,00 / R$ 5.000,00
b) R$ 5.800,00 / R$ 5.000,00
c) R$ 5.000,00 / R$ 5.800,00
d) R$ 5.800,00 / R$ 5.800,00

4. Utilizando como base o enunciado da questão 3, crie um query que satisfaça a descrição.

5. Utilizando como base a tabela Produtos mostrada neste caderno, crie uma atualização, na qual seja possível reajustar o preço dos produtos vendidos em unidades (UN) em 10%. Os outros produtos devem ser reajustados somente em 5%. Esta operação deverá utilizar os comandos de transação, e o desfecho da transação deverá ser automatizado, utilizando a variável @@ERROR.

FINALIZANDO

Neste caderno você conheceu os demais comandos pertencentes à Linguagem de Manipulação de Dados (DML), além de comando de transação e também os comandos da Linguagem de Controle de Dados (DCL).

Através destes comandos, em conjunto com os vistos nos módulos anteriores, você tem a capacidade de programar grandes e complexos bancos de dados.

REFERÊNCIAS

SILBERSCHATZ, Abrahan; KORTH, Henry; SUDARSHAN, S. **Sistema de Banco de Dados**. Elsevier, 2012.

MICROSOFT, **Instruções de Transação** (Transact STL). Disponível em: <http://msdn.microsoft.com/pt-br/library/ms174377.aspx>. Acesso em: 15 abr. 2014.

GLOSSÁRIO

Fórmulas: operações que resultam em um novo valor.

Implícito: de forma escondida, que não precisa ser definido pelo programador.

Base de dados: o mesmo que banco de dados.

Convite à leitura

Neste tema, você terá contato com dois temas avançados relacionados à utilização de bancos de dados: data warehouse (depósito de dados) e data mining (mineração de dados).

Data warehouse é uma forma de se armazenar e centralizar todos os dados de transação sobre um tema específico em um único repositório, estruturado de forma a facilitar análises e consultas.

Data mining é um processo no qual, utilizando-se uma **base de dados**, é possível encontrar padrões que podem ser úteis na geração de conhecimento.

Estas tecnologias são extremamente importantes para o mundo atual, no qual a velocidade e a precisão das informações para o processo de tomada de decisão são primordiais.

Este caderno te auxiliará a entender a relação entre os dois temas. Também será mostrado como o *data warehouse* e o *data mining* se encaixam em um contexto maior, conhecido como extração de conhecimento em base de dados ou, então, *knowledge discovery in database*.

Data Warehouse e *Data Mining*

POR DENTRO DO TEMA

Data Warehouse e *Data Mining*

Com o passar do tempo, a quantidade de dados que passaram a ser armazenados em formato eletrônico aumentou significativamente, e o uso dos sistemas de bancos de dados está diretamente relacionado a esse aumento. Outro fator que contribuiu para que o armazenamento de dados tivesse esse aumento expressivo foi o barateamento do custo de armazenamento ao longo do tempo, bem como a maior disponibilidade de recursos computacionais.

Dados e informações são extremamente importantes para os negócios. Além disso, pode-se dizer que o acesso a estes recursos no momento correto é essencial aos tomadores de decisão.

Os Sistemas de Gerenciamento de Bancos de Dados (SGBDs) que você acompanhou até aqui são responsáveis pela disponibilização de uma parte desses recursos. Os componentes vistos até o momento são excelentes ferramentas para que os dados sejam inseridos nos bancos de dados da maneira mais rápida, simples e segura possível, bem como a extração dos dados armazenados. No entanto, a geração de informações que serão utilizadas para a análise e tomada de decisões é um pouco mais complexa.

Para auxiliar na tarefa da geração de informações de auxílio à tomada de decisões foi criado o conceito de *data warehouse* (depósito de dados). Esse conceito utiliza-se de um banco de dados especialmente preparado para o armazenamento de dados, de forma que a extração de informações utilizadas para análise seja facilitada.

Normalmente, os dados de um *data warehouse* são guardados separadamente dos dados dos sistemas de transação, e esta base de dados especial é utilizada como um repositório centralizado de dados

que, por possuir uma forma mais bem-estruturada, é capaz de fornecer informações a executivos, gerentes, coordenadores e analistas mais rapidamente.

Uma das características principais de um *data warehouse* é que ele deve ser orientado por assuntos. Por exemplo, em um *data warehouse* projetado para concentrar dados referentes às vendas realizadas por uma empresa, perguntas como "quais os produtos mais vendidos?", "quais são os clientes que mais compram?" e "nossos maiores clientes compram quais produtos?" devem ser facilmente respondidas.

A não volatilidade também é uma característica dos dados armazenados em um sistema baseado em *data warehouse*. Uma das missões a ser realizada pelas ferramentas de um *data warehouse* é a possibilidade de análise de dados históricos, e, para isso, a partir do momento em que eles ingressam na base dados que concentra as informações, os dados não podem ser alterados.

A última das características importantes de um *data warehouse* é que as informações armazenadas nele devem ser comparáveis ao longo do tempo. Por exemplo, pode ser necessário saber a variação das vendas de determinado produto durante o ano ou durante uma década. Diferentemente de um banco de dados convencional, no qual, em nome da performance do banco de dados, pode ser necessário remover e armazenar em locais separados os dados históricos, em um *data warehouse* esta informação é vital.

No princípio, os dados de transação eram inseridos no *data warehouse* por meio de instruções de SQL, em um processo considerado bastante complexo. Por serem derivados de vários bancos de dados com informações transacionais, os dados devem passar por uma fase de transformação para que possam se adequar ao formato definido para o armazenamento. Neste processo, pequenas inconsistências são corrigidas, bem como casos em duplicidade. Após a transformação, o *data warehouse* é alimentado com as informações já padronizadas.

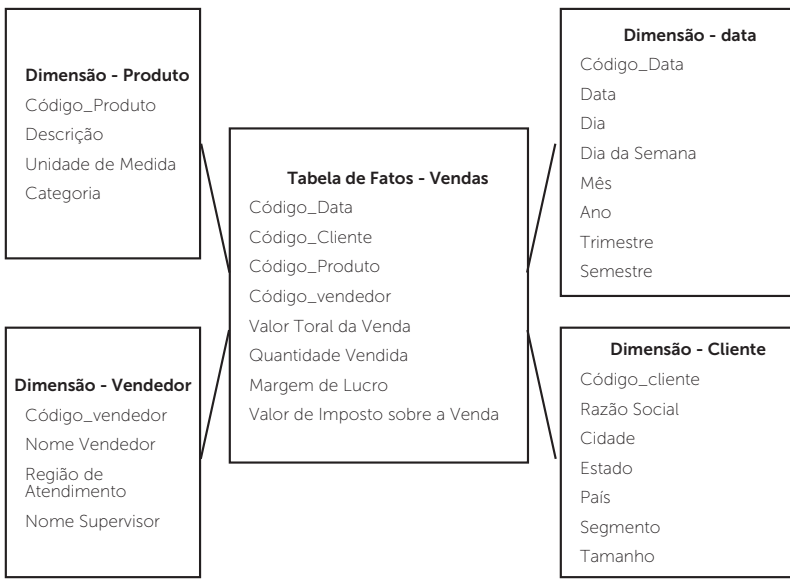
Este conjunto de instruções foi chamado de ETL (Extraction, Transformation e Loading, ou seja, extração, transformação e carregamento). Com o passar do tempo, ferramentas específicas para este fim foram criadas, o que facilitou a tarefa da obtenção dos dados. Essas ferramentas auxiliam todo o processo, passando pelas fases de

extração dos dados, higienização, enriquecimento e transformação dos dados, e carregamento dos dados. Outra tarefa bastante importante é o debug da execução do processo de ETL.

Para que seja realizado o projeto lógico do banco de dados que será utilizado na construção de um *data warehouse*, criou-se a modelagem dimensional. Esta ferramenta utiliza-se de uma grande tabela principal, onde são armazenados os dados (tabela de fatos) e várias tabelas acessórias menores (tabelas de dimensão). Essas tabelas estão relacionadas por chaves, sendo que a tabela de fatos possui uma chave composta que a interliga a cada uma das tabelas de dimensão. Este modelo é conhecido como esquema estrela (star schema) em razão de sua representação gráfica.

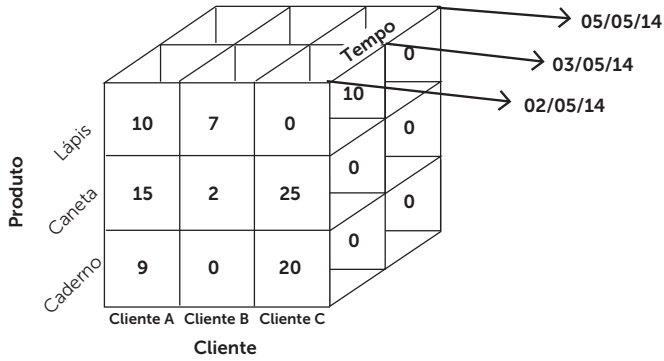
Com base no esquema estrela, os dados são armazenados não normalizados, ou seja, todos os dados de transação são guardados em uma única tabela, sem nenhum tipo de relacionamento entre tabelas de fatos. Esta medida é tomada para que o desempenho do banco de dados durante uma consulta seja melhor e, também, para facilitar seu uso pelo usuário final. Um exemplo de esquema estrela pode ser visto na Figura 4.1:

Figura 4.1 | Diagrama - esquema estrela



Por causa de sua estrutura baseada em fatos e dimensões variando ao longo do tempo, a cada um dos conjuntos de dados dá-se o nome de cubo. Por exemplo, os dados de um cubo de vendas podem relacionar a quantidade de cada produto comprada por cliente variando ao longo do tempo, como pode ser visto na Figura 4.2:

Figura 4.2 | Representação de um cubo



Caso seja necessário comparar o vendedor que efetuou a venda, deve-se adicionar outra dimensão ao modelo mostrado anteriormente. O mesmo processo deve ser realizado caso sejam relevantes informações como a unidade federativa e a cidade da venda, estação do ano, se houve algum tipo de promoção para a venda ou qual loja efetuou a operação. Embora seja impossível representar graficamente este tipo de relação por meio de um simples cubo, seu funcionamento é o mesmo do cubo demonstrado na Figura 4.2.

Levando-se em consideração o conceito de cubo, qualquer informação pode ser extraída ao se fatiar o cubo em qualquer um dos eixos. Por exemplo, pode-se obter o número de lápis vendidos para cada cliente ao longo do tempo ou, então, é possível saber quantos produtos foram vendidos para cada cliente no dia 03/05/14.

Para facilitar este tipo de análise, é utilizada uma ferramenta conhecida como OLAP (Online Analytical Processing), especialmente projetada para trabalhar com data warehouse.

Utilizando a ferramenta OLAP, é possível fatiar estes dados de forma intuitiva, além de utilizar os recursos de *drill up* e *drill down*.

Drill up e drill down são formas de se navegar em dados que estão agrupados. Pode ser considerado um exemplo de dados agrupados o local para onde os produtos são vendidos, sendo o topo da pilha o país onde foi feita a venda, seguido por região, unidade federativa e cidade. Ao descer na granularidade, por exemplo, estar visualizando o nível de unidade federativa e descer para o nível de cidade, será realizada a operação de *drill down*. Caso seja o contrário, e o nível esteja subindo, a operação é chamada de *drill up*.

Por meio dessas operações, as análises gerenciais podem ser feitas em tempo muito menor do que se fossem realizadas análises convencionais, e a tomada de decisão por parte de gestores pode ser realizada com maior precisão e agilidade - fatores indispensáveis para o mundo competitivo atual.

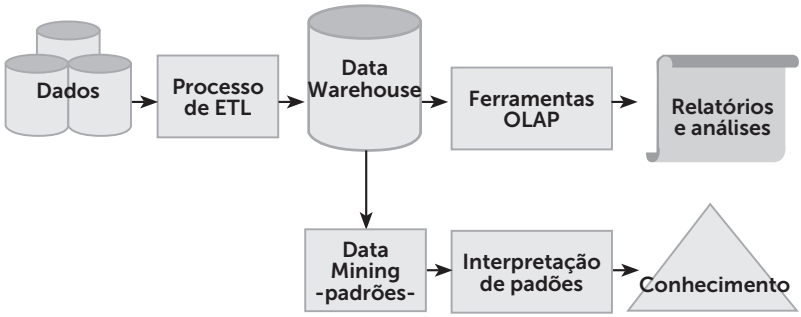
A base de dados centralizada gerada pelo data warehouse pode ser utilizada também para a realização de data mining (mineração de dados). Este conceito consiste em encontrar padrões em bases de dados, por meio de inteligência artificial, redes neurais e métodos estatísticos. Todo este esforço é realizado para que seja possível obter conhecimento no meio de uma grande massa de dados, muitas vezes, desconexos.

Um exemplo de padrão fundamentado em associações que pode ser identificado e que faz parte do cotidiano são as sugestões de compras em sites. São indicados produtos similares aos que você comprou ou visitou nos últimos dias ou, então, são identificadas pessoas que têm o mesmo perfil de compras para indicar produtos comprados por elas.

Previsões também podem ser utilizadas a partir de um sistema de data mining, por exemplo, uma seguradora pode prever a chance de um candidato a uma nova apólice de seguros ter algum sinistro, com base nas informações conhecidas, tais como faixa etária, região, modelo do veículo, distância dirigida por semana e sinistros anteriores.

A identificação de um padrão é parte de um contexto maior, conhecido como extração de conhecimento em base de dados ou, então, *Knowledge Discovery in Database* (KDD). O KDD filtra e refina os padrões identificados pela operação de data mining, fazendo com que sejam utilizados somente os padrões válidos e que façam sentido para a tomada de decisão necessária. Durante o processo, várias etapas de ajuste podem realizadas nos dados e na forma da mineração de dados escolhida, a fim de obter melhores resultados para o processo. A relação entre data warehouse, *data mining* e *knowledge discovery in database* pode ser vista na Figura 4.3.

Figura 4.3 | *Knowledge discovery in database*



ACOMPANHE NA WEB

BI e *Data Warehouse*

- Acesse no site Kimball Group a seção sobre BI e *Data Warehouse*. Disponível em: <https://goo.gl/733etZ/>. Acesso em: 1 junho 2014.

A Dimensional Modeling Manifesto

- Ralph Kimball, fundador do Kimball Group, foi um dos pioneiros na definição de conceitos relacionados a *Data Warehouse*. Este site está repleto de artigos importantes, tal como o manifesto sobre a modelagem dimensional. KIMBALL, R. *A Dimensional Modeling Manifesto*. Disponível em: <https://goo.gl/0xvnM6/>. Acesso em: 1 jun. 2014.

AGORA É A SUA VEZ

Instruções:
Agora, chegou a sua vez de exercitar seu aprendizado. A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

1. Dentro do contexto de data mining, defina o que é um "padrão". Dê um exemplo de padrão e uma decisão que possa ser tomada baseada nesse padrão.

2. A ferramenta utilizada para realizar análise de dados não normalizados recebe o nome de:
a) OLTP (Online Transaction Processing).
b) MIS (Management Information System).
c) KDD (Knowledge Discovery in Database).
d) SGBD (Sistema Gerenciado de Banco de Dados).
e) N.D.A.

3. Classifique como drill up (u) ou drill down (d) cada uma das alterações de ponto de vista a seguir:
a) ( ) De [unidade federativa] para [cidade].
b) ( ) De [modelo do veículo] para [fabricante].
c) ( ) De [funcionário] para [setor].
d) ( ) De [mês do ano] para [dias do mês].
e) ( ) De [apartamento] para [prédio].

4. Explique cada um dos elementos do processo de ETL.

5. Qual o relacionamento existente entre data mining e data warehouse? Como eles se relacionam com o knowledge discovery in database?

FINALIZANDO

Nesse tema, você teve a oportunidade de conhecer os conceitos básicos de temas bastante complexos – *data warehouse* e *data mining*. Cada um desses temas merece uma maior atenção, por isso vale a pena procurar materiais complementares para enriquecer seu

conhecimento acerca do assunto.

A correta aplicação destes métodos pode acelerar a obtenção de informações relevantes e conhecimentos para a tomada de decisões.

REFERÊNCIAS

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. **Sistema de Banco de Dados**. São Paulo: Elsevier, 2012.

GLOSSÁRIO

Base de dados: o mesmo que banco de dados.

Granularidade: o menor nível de informação disponível para acesso.

Redes Neurais: modelo computacional que simula o sistema nervoso central, para que seja possível o aprendizado de máquina.

ISBN 978-85-8482-867-8