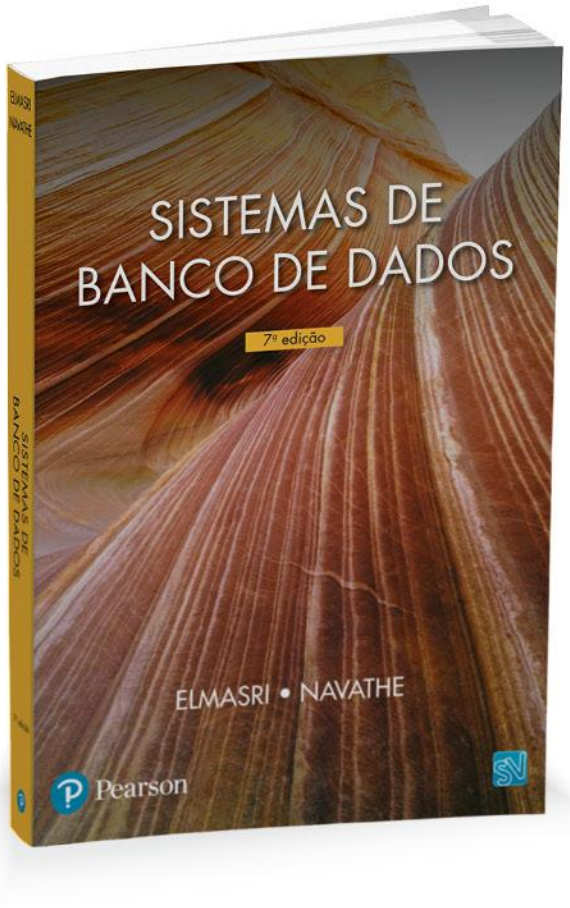
Bancos de Dados de Objeto e Objeto-Relacional

Slides: Leonardo Chaves Dutra da Rocha

lcrocha@ufsj.edu.br

Bancos de dados distribuídos

Bancos de dados NOSQL e sistemas de armazenamento Big Data



Bancos de dados de objeto e objeto-relacional

- **Banco de dados de objeto (BDO)**
  - **Sistemas de gerenciamento de dados de objeto (SGDO)**
  - Foram propostos para atender a algumas das necessidades de aplicações mais complexas
  - Especifica:
    - A estrutura dos objetos complexos
    - As operações que podem ser aplicadas a esses objetos

Visão geral dos conceitos de banco de dados de objeto

- Introdução aos conceitos e recursos orientados a objeto
  - Origens nas linguagens de programação OO
  - Um objeto possui dois componentes:
    - Estado (valor) e comportamento (operações)
  - Variáveis de instância
    - Mantém os valores que definem o estado interno do objeto
  - Operação definida em duas partes:
    - Assinatura ou interface e implementação

Visão geral dos conceitos de banco de dados de objeto

- Herança
  - Permite a especificação de novos tipos ou classes que herdam grande parte de sua estrutura e/ou operações de tipos ou classes previamente definidas
- Sobrecarga de operador
  - Capacidade de uma operação de ser aplicada a diferentes tipos de objetos
  - Um nome de operação pode se referir a várias implementações distintas

Identidade de objeto e objetos *versus* literais

- Identidade única
  - Implementada por meio de um identificador de objeto (OID) único
  - **Imutável**
  - A maioria dos sistemas de banco de dados OO permite a representação de objetos e literais (ou valores)

Estruturas de tipo complexas para objetos e literais

- Estrutura de complexidade arbitrária
  - Contém todas as informações necessárias que descrevem o objeto ou literal
- Aninhamento de **construtores de tipos**
  - Tipo complexo pode ser construído com base em outros tipos
- Construtores mais básicos:
  - **Átomo:** contém tipos básicos (inteiro, real, etc.);
  - **Struct (ou tupla):** gerador de tipos;
  - **Coleção:** para aninhar estruturas mais complexas;

Estruturas de tipo complexas para objetos e literais

- Tipos de coleção:
  - **Set**
  - **List**
  - **Bag**
  - **Array**
  - **Dictionary**

Estruturas de tipo complexas para objetos e literais

```
define type  FUNCIONARIO
tuple (
    Pnome:                  string;
    Minicial:               char;
    Unome:                  string;
    Cpf:                    string;
    Data_nascimento:        DATE;
    Endereco:               string;
    Sexo:                   char;
    Salario:                float;
    Supervisor:             FUNCIONARIO;
    Dep:                    DEPARTAMENTO;

define type  DATA
tuple (
    Ano:                    integer;
    Mes:                    integer;
    Dia:                    integer; );

define type  DEPARTAMENTO
tuple (
    Dnome:                  string;
    Dnumero:                integer;
    Ger:                    tuple (
                            Gerente: FUNCIONARIO;
                            Data_inicio: DATE; );
    Localizacoes:           set(string);
    Funcionarios:           set(FUNCIONARIO);
    Projetos:               set(PROJETO); );
```

Figura 11.1
Especificando os tipos de objeto FUNCIONARIO, DATA e DEPARTAMENTO usando construtores de tipo.

Encapsulamento de operações e persistência de objetos

- Encapsulamento
  - Relacionado aos conceitos de tipos de dados abstratos e ocultação de informações nas linguagens de programação
  - Define o comportamento de um tipo de objeto com base nas operações que podem ser aplicadas externamente
  - Os usuários externos só se tornam cientes da interface das operações
  - Divide a estrutura de um objeto em atributos visíveis e ocultos

Encapsulamento de operações

- **Construtor de objeto**
  - Usado para criar um objeto
- **Operação de destruição**
  - Usado para destruir (excluir) um objeto
- **Operações modificadoras**
  - Modificar os estados (valores) de vários atributos de um objeto
- **Recupera** informações sobre o objeto
- Uma operação costuma ser aplicada a um objeto usando a notação de ponto

Persistência de objetos

- **Objetos transientes**
  - Existem no programa em execução
  - Desaparecem quando o programa termina
- **Objetos persistentes**
  - Armazenados no banco de dados e persistem após o término do programa
  - **Mecanismo de nomeação**
  - **Acessibilidade**

Persistência de objetos

```
define class SET_DEPARTAMENTO
    type set (DEPARTAMENTO);
    operations   adiciona_dep(d: DEPARTAMENTO):   boolean;
                 (* acrescenta um departamento ao objeto SET_DEPARTAMENTO *)
                 remove_dep(d: DEPARTAMENTO): boolean;
                 (* remove um departamento do objeto SET_DEPARTAMENTO *)
                 criar_set_dep: SET_DEPARTAMENTO;
                 destroi_set_dep: boolean;
end SET_DEPARTAMENTO;
...
persistent name TODOS_DEPARTAMENTOS: SET_DEPARTAMENTO;
(* TODOS_DEPARTAMENTOS é um objeto persistente nomeado do tipo SET_DEPARTA-
MENTO *)
...
d:= cria_dep;
(* cria um objeto DEPARTAMENTO na variável d *)
...
b:= TODOS_DEPARTAMENTOS.adiciona_dep(d);
(* torna d persistente incluindo-o no conjunto persistente TODOS_DEPARTAMENTOS *)
```

Figura 11.3
Criando objetos persistentes por nomeação e acessibilidade.

Hierarquias de tipo e herança

- Herança
  - Definição de novos tipos com base em outros predefinidos
  - **hierarquia de tipo** (ou **classe**)

Hierarquias de tipo e herança

- **Subtipo**
  - Útil na criação de um novo tipo que é similar, mas não idêntico a um tipo já definido
  - Exemplo:
    - `FUNCIONÁRIO subtype-of PESSOA: Salário, Data_contratação, Nível`
    - `ALUNO subtype-of PESSOA: Curso, Coeficiente`

Outros conceitos de orientação a objeto

- **Poliformismo** de operações
  - Também conhecido como **sobrecarga de operador**
  - Permite que o mesmo nome de operador ou símbolo esteja ligado a duas ou mais implementações diferentes
  - Dependendo do tipo de objetos aos quais o operador é aplicado

Outros conceitos de orientação a objeto

- **Herança múltipla**
  - Subtipo herda funções (atributos e métodos) de mais de um supertipo
- **Herança seletiva**
  - Subtipo herda apenas algumas das funções de um supertipo

Recursos objeto-relacional: extensões do banco de dados de objeto para SQL

- **Construtores de tipo**
  - Especifica objetos complexos
- Mecanismo para especificar a **identidade de objeto**
- **Encapsulamento de operações**
  - Fornecido por meio do mecanismo de tipos definidos pelo usuário (UDTs)
- Mecanismos de **herança**
  - Fornecidos usando palavra-chave `UNDER`

Tipos definidos pelo usuário e estruturas complexas para objetos

```
(a) CREATE TYPE TIPO_END_RUA AS (
        NUMERO              VARCHAR (5),
        NOME_RUA            VARCHAR (25),
        NR_APTO             VARCHAR (5),
        NR_BLOCO            VARCHAR (5)
    );
    CREATE TYPE TIPO_END_BRASIL AS (
        END_RUA             TIPO_END_RUA,
        CIDADE              VARCHAR (25),
        CEP                 VARCHAR (10)
    );
    CREATE TYPE TIPO_TELEFONE_BRASIL AS (
        TIPO_TELEFONE       VARCHAR (5),
        CODIGO_AREA         CHAR (3),
        NUM_TELEFONE        CHAR (7)
    );

(b) CREATE TYPE TIPO_PESSOA AS (
        NOME                VARCHAR (35),
        SEXO                CHAR,
        DATA_NASCIMENTO     DATE,
        TELEFONES           TIPO_TELEFONE_BRASIL ARRAY [4],
        END                 TIPO_END_BRASIL
    INSTANTIABLE
    NOT FINAL
    REF IS SYSTEM GENERATED
    INSTANCE METHOD IDADE( ) RETURNS INTEGER;
    CREATE INSTANCE METHOD IDADE( ) RETURNS INTEGER
        FOR TIPO_PESSOA
        BEGIN
            RETURN /* CÓDIGO PARA CALCULAR A IDADE DE UMA PESSOA COM BASE NA
                      DATA DE HOJE E SUA DATA_NASCIMENTO */
        END;
    );

(c) CREATE TYPE TIPO_NOTA AS (
        DISCIPLINA          CHAR (8),
        SEMESTRE            VARCHAR (8),
        ANO                 CHAR (4),
        NOTA                CHAR
    );
    CREATE TYPE TIPO_ALUNO UNDER TIPO_PESSOA AS (
        CODIGO_CURSO        CHAR (4),
        COD_ALUNO           CHAR (12),
        GRAU                VARCHAR (5),
        HISTORICO_ESCOLAR   TIPO_NOTA ARRAY [100]
```

(continua)

Figura 11.4
Ilustrando alguns dos recursos de objeto da SQL. (a) Usando UDTs como tipos para atributos como Endereco e Telefone. (b) Especificando UDT para TIPO_PESSOA. (c) Especificando UDTs para TIPO_ALUNO e TIPO_FUNCIONARIO como dois subtipos de TIPO_PESSOA.

```
    INSTANTIABLE
    NOT FINAL
    INSTANCE METHOD COEFICIENTE( ) RETURNS FLOAT;
    CREATE INSTANCE METHOD COEFICIENTE( ) RETURNS FLOAT
        FOR TIPO_ALUNO
        BEGIN
            RETURN /* CÓDIGO PARA CALCULAR COEFICIENTE MEDIO DE UM ALUNO COM BASE EM
                      SEU HISTORICO ESCOLAR */
        END;
    );
    CREATE TYPE TIPO_FUNCIONARIO UNDER TIPO_PESSOA AS (
        CODIGO_EMPREGO      CHAR (4),
        SALARIO             FLOAT,
        CPF                 CHAR (11)
    INSTANTIABLE
    NOT FINAL
    );
    CREATE TYPE TIPO_GERENTE UNDER TIPO_FUNCIONARIO AS (
        DEP_GERENCIADO      CHAR (20)
    INSTANTIABLE
    );

(d) CREATE TABLE PESSOA OF TIPO_PESSOA
        REF IS ID_PESSOA SYSTEM GENERATED;
    CREATE TABLE FUNCIONARIO OF TIPO_FUNCIONARIO
        UNDER PESSOA;
    CREATE TABLE GERENTE OF TIPO_GERENTE
        UNDER FUNCIONARIO;
    CREATE TABLE ALUNO OF TIPO_ALUNO
        UNDER PESSOA;

(e) CREATE TYPE TIPO_EMPRESA AS (
        NOME_EMP            VARCHAR (20),
        LOCALIZACAO         VARCHAR (20));
    CREATE TYPE TIPO_EMPREGO AS (
        Funcionario REF (TIPO_FUNCIONARIO) SCOPE (FUNCIONARIO),
        Empresa REF (TIPO_EMPRESA) SCOPE (EMPRESA) );
    CREATE TABLE EMPRESA OF TIPO_EMPRESA (
        REF IS COD_EMP SYSTEM GENERATED,
        PRIMARY KEY (NOME_EMP) );
    CREATE TABLE EMPREGO OF TIPO_EMPREGO;
```

Figura 11.4 (continuação)
Ilustrando alguns dos recursos de objeto da SQL. (c) (continuação) Especificando UDTs para TIPO_ALUNO e TIPO_FUNCIONARIO como dois subtipos de TIPO_PESSOA. (d) Criando tabelas com base em alguns dos UDTs, e ilustrando a herança de tabela. (e) Especificando relacionamentos com REF e SCOPE.

Criando tabelas baseadas nos UDTs

- **INSTANTIABLE**
  - Especifica que a UDT é instanciável
  - Faz com que uma ou mais tabelas sejam criadas

Encapsulamento de operações

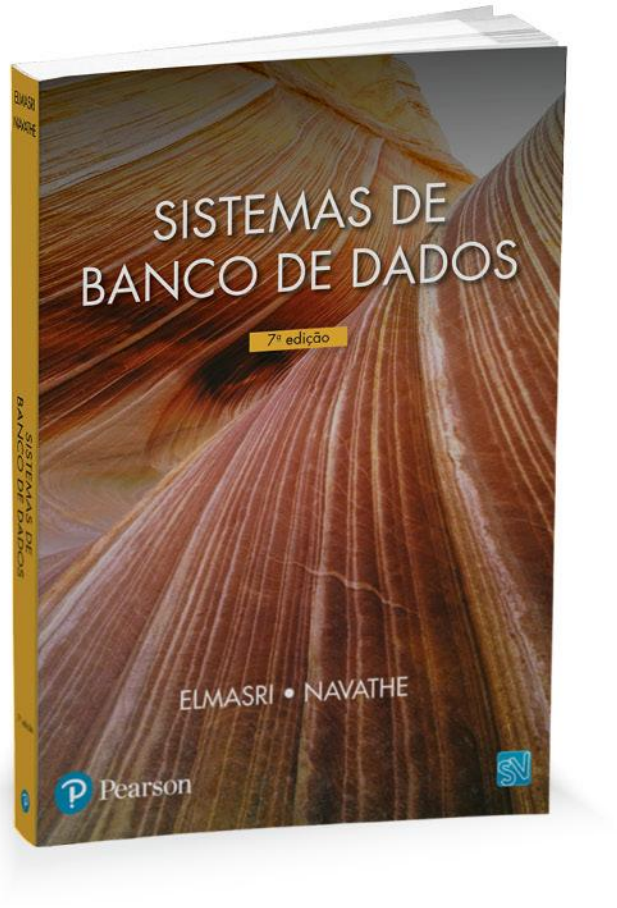
- Tipo definido pelo usuário
  - Especifica métodos (ou operações) além dos atributos
  - Formato:

```
CREATE TYPE <NOME-TIPO> (
<LISTA DE ATRIBUTOS DE COMPONENTE E
SEUS TIPOS>
<DECLARACAO DE FUNcoES (METODOS)>
);
```

Especificando relacionamentos por referência

- Um atributo componente de uma tupla pode ser uma **referência** a uma tupla de outra tabela
  - Especificada usando a palavra-chave **`REF`**
- Palavra-chave **`SCOPE`**
  - Especifica o nome da tabela cujas tuplas podem ser referenciadas

Bancos de dados distribuídos



Constituição de um BDD

- Para um banco de dados ser chamado de distribuído, as seguintes condições mínimas devem ser satisfeitas:

1. Conexões de nós de banco de dados por uma rede de computadores.

2. Inter-relação lógica dos bancos de dados conectados.

3. Possível ausência de homogeneidade entre os nós conectados.

- As redes podem ter diferentes **topologias** que definem os caminhos de comunicação entre os nós.

- O que importa é que cada nó seja capaz de se comunicar, direta ou indiretamente, com todos os outros nós.

Transparência

SISTEMAS DE BANCO DE DADOS
7ª edição
ELMASRI • NAVATHE

- O conceito de **transparência** estende a ideia geral de ocultar detalhes da implementação dos usuários finais.

- Os seguintes tipos de transparência são possíveis:

1. Transparência da organização dos dados (também conhecida como transparência de distribuição ou de rede).

2. Transparência de replicação.

3. Transparência de fragmentação.

4. Outras transparências incluem transparência de projeto e transparência de execução.

Confiabilidade e disponibilidade

- A **confiabilidade** é definida em termos gerais como a probabilidade de um sistema estar funcionando (não parado) em certo ponto no tempo.
- A **disponibilidade** é a probabilidade de que o sistema esteja continuamente disponível durante um intervalo de tempo.
- Uma **falha** pode ser descrita como um desvio de um comportamento do sistema daquele especificado a fim de garantir a execução correta das operações.
- **Erros** constituem o subconjunto de estados do sistema que causam a falha.
- **Defeito** é a causa de um erro.

Escalabilidade e tolerância à partição

- A **escalabilidade** determina a extensão à qual o sistema pode expandir sua capacidade e continuar operando sem interrupção.

- Existem dois tipos de escalabilidade:

1. Escalabilidade horizontal

2. Escalabilidade vertical

- A **autonomia** determina a extensão à qual os nós individuais ou BDs em um BDD conectado podem operar independentemente.

- Algumas vantagens importantes dos BDDs são listadas a seguir:

Escalabilidade e tolerância à partição

SISTEMAS DE BANCO DE DADOS
7ª edição
ELMASRI • NAVATHE

1. Maiores facilidade e flexibilidade de desenvolvimento da aplicação.
2. Maior disponibilidade
3. Melhor desempenho
4. Facilidade de expansão por meio da escalabilidade

Fragmentação de dados e sharding

- Uma técnica, chamada **fragmentação horizontal** ou **sharding**, pode ser usada para particionar cada relação por departamento, por exemplo.

Fragmentação de dados e sharding

- Um **fragmento horizontal** ou **shard** de uma relação é um subconjunto das tuplas nessa relação.
- A **fragmentação horizontal derivada** se aplica ao particionamento de uma relação primária para outras relações secundárias, que estão relacionadas à primária por meio de uma chave estrangeira.
- Desse modo, os dados relacionados entre as relações primária e secundária são fragmentados da mesma maneira.
- A **fragmentação vertical** divide uma relação "verticalmente", por colunas.
- Um **fragmento vertical** de uma relação mantém apenas certos atributos da relação.

Fragmentação de dados e sharding

- Podemos misturar os dois tipos de fragmentação, produzindo uma **fragmentação mista**.

- Um **esquema de fragmentação de um banco de dados** é uma definição de um conjunto de fragmentos que inclui todos os atributos e tuplas no banco de dados e satisfaz a condição de que o banco de dados inteiro pode ser reconstruído com base nos fragmentos ao aplicar alguma sequência de operações de junção.

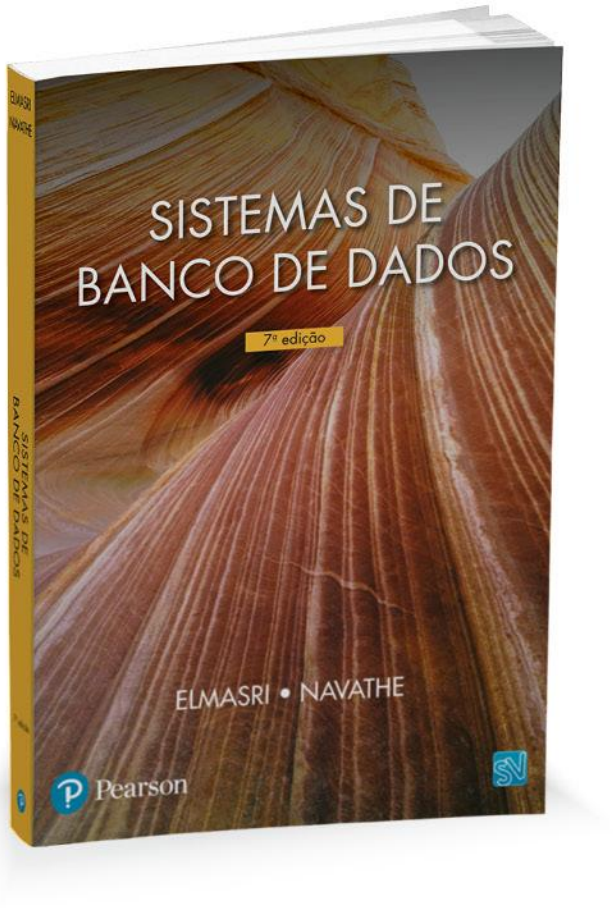
- Um **esquema de alocação** descreve a alocação de fragmentos aos nós (sites) do SBDD.

- Logo, esse é um mapeamento que especifica para cada fragmentação o(s) nó(s) em que ela está armazenada.

Replicação e alocação de dados

- O caso mais extremo é a replicação do banco de dados inteiro em cada nó no sistema distribuído, criando assim um **banco de dados distribuído totalmente replicado**.
- Isso pode melhorar bastante a disponibilidade porque o sistema continua a operar desde que pelo menos um nó esteja funcionando.
- O outro extremo da replicação total envolve **não ter replicação**.
- Nesse caso, todos os fragmentos precisam ser disjuntos, exceto pela repetição das chaves primárias entre os fragmentos verticais.
- Isso também é chamado de **alocação não redundante**.
- Entre esses dois extremos, temos uma grande variedade de **replicação parcial dos dados**.

Bancos de dados NOSQL e sistemas de armazenamento Big Data



Surgimento de sistemas NOSQL

- Algumas das organizações que depararam com aplicações de gerenciamento e armazenamento de dados decidiram desenvolver seus próprios sistemas:

- A Google desenvolveu um sistema NOSQL proprietário conhecido como BigTable, que é usado em muitas aplicações da empresa que exigem grandes quantidades de armazenamento de dados, como o Gmail, o Google Maps e o buscador (máquina de busca).

- A Amazon desenvolveu um sistema NOSQL chamado DynamoDB, que está disponível por meio dos serviços em nuvem da Amazon.

- O Facebook desenvolveu um sistema NOSQL chamado Cassandra, que agora é de código aberto e conhecido como Apache Cassandra.

Características dos sistemas NOSQL

- Características do NOSQL relacionadas a bancos de dados distribuídos e sistemas distribuídos:

1. Escalabilidade
2. Disponibilidade, replicação e consistência eventual
3. Modelos de replicação
4. Sharding de arquivos
5. Acesso a dados com alto desempenho

Características dos sistemas NOSQL

- Características do NOSQL relacionadas a modelo de dados e linguagens de consulta:

1. Não exigência de um esquema
2. Linguagens de consulta menos poderosas
3. Versionamento

- A categorização mais comum lista as quatro principais categorias a seguir:

1. Sistemas NOSQL baseados em documentos
2. Armazenamentos de chave-valor do NOSQL
3. Sistemas NOSQL baseados em coluna ou em largura de coluna
4. Sistemas NOSQL baseados em grafos

O teorema CAP

- O **teorema CAP**, introduzido originalmente como o princípio CAP, pode ser usado para explicar alguns dos requisitos concorrentes em um sistema distribuído com replicação.

- As três letras no CAP referem-se a três propriedades desejáveis de sistemas distribuídos com dados replicados:

1. **consistência** (**consistency** — consistência entre as cópias replicadas),

2. **disponibilidade** (**availability** — disponibilidade do sistema para operações de leitura e gravação) e

3. **tolerância à partição** (**partition tolerance** — tolerância à partição, em face de os nós no sistema estarem sendo particionados por uma falha na rede).

Sistemas NOSQL baseados em documentos e MongoDB

- Sistemas NOSQL baseados em documentos ou orientados a documento geralmente armazenam dados como **coleções de documentos** semelhantes.
- Esses tipos de sistemas também são conhecidos como **armazenamentos de documentos**.
- O sistema basicamente extrai os nomes dos elementos de dados dos documentos autodescritivos na coleção.
- **Documentos** individuais são armazenados em uma **coleção**.
- Uma coleção não tem um esquema.

Sistemas NOSQL baseados em documentos e MongoDB

SISTEMAS DE BANCO DE DADOS
7ª edição
ELMASRI • NAVATHE

- Exemplo de documentos simples em MongoDB. Projeto de documento desnormalizado com subdocumentos embutidos:

```
{
    _id:                "P1",
    Nome_projeto:       "ProdutoX",
    Local_projeto: "São Paulo",
    Trabalhadores: [
                        { Nome_func: "João Silva",
                        Horas: 32.5
                        },
                        { Nome_func: "Joice Leite",
                        Horas: 20.0
                         }
                    ]
};
```

Sistemas NOSQL baseados em documentos e MongoDB

SISTEMAS DE BANCO DE DADOS
7ª edição
ELMASRI • NAVATHE

- Exemplo de documentos simples em MongoDB. Vetor embutido de referências de documento:

```
{
    _id:              "P1",
    Nome_projeto:     "ProdutoX",
    Local_projeto:    "São Paulo",
    IdsTrabalhador:   [ "T1", "T2" ]
}
    { _id:            "T1",
    Nome_func:        "João Silva",
    Horas:            32.5
}
    { _id:            "T2",
    Nome_func:        "Joice Leite",
    Horas:            20.0
}
```

Sistemas NOSQL baseados em documentos e MongoDB

SISTEMAS DE BANCO DE DADOS
7ª edição
ELMASRI • NAVATHE

- Exemplo de documentos simples em MongoDB. Documentos normalizados:

```
{
    _id:              "P1",
    Nome_projeto:     "ProdutoX",
    Local_projeto:    "São Paulo"
}
{   _id:              "T1",
    Nome_func:        "João Silva",
    IdProjeto:        "P1",
    Horas:            32.5
}
{   _id:              "T2",
        Nome_func:    "Joice Leite",
        IdProjeto:    "P1",
        Horas:        20.0
}
```

Sistemas NOSQL baseados em documentos e MongoDB

SISTEMAS DE BANCO DE DADOS
7ª edição
ELMASRI • NAVATHE

- Exemplo de documentos simples em MongoDB. Inserindo os documentos da figura anterior em suas coleções:

```
db.projeto.insert( { _id: "P1", Nome_projeto: "ProdutoX", Local_projeto: "São Paulo" } )
db.trabalhador.insert( [ { _id: "T1", Nome_func: "João Silva", IdProjeto: "P1", Horas:
                          32.5 },
    { _id: "T2", Nome_func: "Joice Leite", IdProjeto: "P1", Horas: 20.0 } ] )
```

Sistemas NOSQL baseados em documentos e MongoDB

- MongoDb possui diversas operações CRUD, em que CRUD significa criar, ler, atualizar, excluir (create, read, update, delete).

- Os documentos podem ser criados e inseridos em suas coleções usando a operação insert, cujo formato é:

db.<nome_coleção>.insert(<documento(s)>)

- A operação de exclusão é chamada de remove, e seu formato é:

db.<nome_coleção>.remove(<condição>)

- Para consultas de leitura, o comando principal é chamado find, e seu formato é:

db.<nome_coleção>.find(<condição>)

Armazenamentos chave-valor em NOSQL

- Os **armazenamentos chave-valor** se concentram no alto desempenho, disponibilidade e escalabilidade, armazenando dados em um sistema de armazenamento distribuído.

- A **chave** é um identificador único associado a um item de dados e é usada para localizar esse item de dados rapidamente.

- O **valor** é o próprio item de dados e pode ter formatos muito diferentes para diferentes sistemas de armazenamento chave-valor.

- A principal característica dos armazenamentos chave-valor é o fato de que cada valor (item de dados) precisa estar associado a uma chave única e que a recuperação do valor, ao fornecer uma chave, deve ser muito rápida.

Sistemas NOSQL baseados em coluna ou em largura de coluna

- Outra categoria de sistemas NOSQL é conhecida como **sistemas baseados em coluna** ou em **largura de coluna**.
- BigTable (e Hbase) às vezes é descrito como um mapa ordenado persistente distribuído multidimensional esparso, em que a palavra mapa significa uma coleção de pares (chave, valor) (a chave é mapeada para o valor).
- Uma das principais diferenças que distinguem os sistemas baseados em colunas dos armazenamentos chave-valor é a natureza da chave.
- Em sistemas baseados em coluna, como o Hbase, a chave é multidimensional e, portanto, possui vários componentes: geralmente, uma combinação de nome de tabela, chave de linha, coluna e rótulo de data e hora (timestamp).