Bancos de Dados Não-Convencionais

- BDs Objeto-Relacionais
- BDs Orientados a Objetos
- BDs XML

BDs Não-Convencionais

- BDs relacionais são os mais usados hoje em dia
  - Fáceis de usar e de modelar
  - Consolidados no mercado
  - Bom desempenho e escalabilidade
- BDs relacionais possuem limitações
  - Poucos tipos de dados
  - Dado geralmente pequeno, com tamanho predefinido
  - Valor de um dado é atômico (1ª forma normal)
  - São inapropriados para armazenamento e busca de novos tipos de dados como imagens, áudio e vídeo

BDs Não-Convencionais

- BDs relacionais não são adequados para modelar os dados de várias aplicações surgidas nos últimos anos
  - Multimídia
  - Hipertexto
  - Geoprocessamento
  - Ensino à distância
  - Medicina
  - CRM
  - CAD
  - ...

BDs Não-Convencionais

- Soluções propostas
  - Modificar o modelo relacional
    - BDs Objeto-Relacionais
  - Adotar novos modelos
    - BDs Orientados a Objetos
    - BDs XML
    - BDs Temporais
    - BDs Geográficos
    - ...

BDs Objeto-Relacionais

- BDs Objeto-Relacionais (OR)
  - Adicionam a abstração de objetos no modelo relacional, tornando o modelo de dados mais rico
  - Suportam novos tipos de dados
  - Mantém a eficiência do modelo relacional para gerencimento de dados
  - Exemplos de BDs Objeto-Relacionais:
    - Oracle
    - IBM DB2
    - Postgres
    - ...

BDs Objeto-Relacionais

- SQL 99 (SQL3)
  - Padrão de linguagem objeto-relacional (em definição)
  - Ainda não é suportada por muitos BDs comerciais
  - Permite a construção de tipos com **create type**
    Ex.: **create type** Pessoa
    (Nome **varchar**, CPF **integer**)
    Obs.: o nome é referenciado como Pessoa.Nome
  - Suporta herança simples, *arrays*, objetos binários (**Blobs**) e de caracteres (**Clobs**)
    Ex.: **create type** Aluno **under** Pessoa
    (Matrícula **integer**, Curso **varchar**
    Disciplinas **varchar array[10]**,
    Foto3x4 **Blob**(32Kb), Histórico **Clob**(16Kb))
    Obs.: elementos do *array* são acessados com [*n*]

BDs Objeto-Relacionais

- SQL 99 (cont.)
  - Podemos criar tabelas e sub-tabelas
    Ex.: **create table** Alunos **of** Aluno
    create table Professores **of** Pessoa
    create table Bolsistas **of** Aluno **under** Alunos
  - *Arrays* são inseridos usando **array**[…]
    Ex.: **insert into** Alunos **values**
    (('João', 12345678900, 01136400,'SIN'),
    20021, **array**['INE5612','INE5613'],
    '10110…', 'Histórico…')
  - Referências apontam para objetos
    Ex.: **create table** Disciplina (Nome **varchar**,
    Professor **ref**(Pessoa) **scope** Professores,
    Turma **ref**(Aluno) **array**[50] **scope** Alunos)

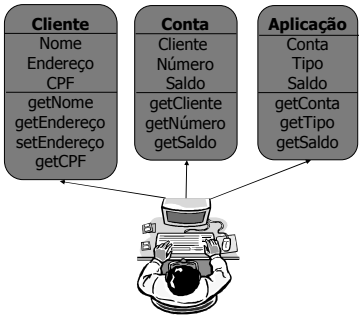
BDs Objeto-Relacionais

- SQL 99 (cont.)
  - Referências são adicionadas explicitamente à tabela
    Ex.: **create table** Alunos **of** Aluno
    ref is oid **system generated**
  - Referências são obtidas como atributos da tabela
    Ex.: **select** oid **from** Alunos **where** Nome = `João'
    Obs.: Oracle e alguns outros BDs usam **ref**(*obj*)
  - Métodos manipulam os dados
    Ex.: **create method** TransfInterna (matric **integer**,
    curso varchar) **for** Aluno
    begin
    set self.Matrícula = matric;
    set self.Curso = curso;
    end

BDs Orientados a Objetos

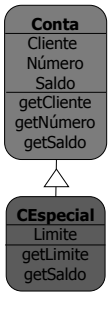
- Dados são modelados como objetos
  - Um Objeto equivale a uma entidade do modelo E-R
  - Objetos encapsulam um conjunto de Variáveis que contém os dados do objeto
  - Métodos são usados para acessar e alterar os dados
  - Mensagens são trocadas entre objetos para ativar seus métodos
    - Carregam os Parâmetros de chamada do método
    - Geram uma Resposta com o retorno do método
    - O envio de mensagens não implica na transmissão de dados pela rede (pode ser implementado como uma chamada de método local)

BDs Orientados a Objetos



BDs Orientados a Objetos

- A classe de um objeto define a sua interface externa
  - Variáveis, métodos e mensagens aceitas
- Podem existir relações de herança entre classes
  - Herança de interface e implementação
  - Ex: Conta → Conta Especial (com limite)
- Métodos podem ter implementações diferentes nas subclasses
  - Ex: getSaldo() da conta especial pode incluir o limite de crédito no cálculo do saldo disponível



BDs Orientados a Objetos

- Identidade
  - Objetos são identificados univocamente no sistema, sem usar para tal os valores dos dados
  - Já em BDs relacionais, a identidade de uma tupla é dada pelo valor de um dado (ex.: chave primária)
- Referências
  - Objeto podem ter referências para outros objetos
    - Ou seja, uma variável de um objeto podem ter como valor a identidade de um outro objeto
    - No exemplo anterior, Conta → Cliente
  - Em BDs relacionais, uma tupla precisa conter um valor chave de outra tupla e usá-lo para localizá-la

BDs Orientados a Objetos

- Diferentes estratégias podem ser usadas para acessar BDs orientados a objeto:
  - Usar orientação a objeto somente no projeto e acessá-lo como um BD relacional
  - Usar uma linguagem semelhante a SQL com extensões para acesso a objetos
  - Usar uma linguagem OO própria do BD
  - Usar uma linguagem OO (C++, Java, …) com extensões para suporte a persistência de dados

13

BDs Orientados a Objetos

- Linguagens definidas pela ODMG
  - ODL (*Object Definition Language*)
    - Permite que sejam definidos dados e objetos persistentes em uma LPOO
  - OML (*Object Manipulation Language*)
    - Fornece mecanismos para que LPOOs manipularem bancos de dados persistentes criados em ODL
  - OQL (*Object Query Language*)
    - Permite que LPOOs consultem bancos de dados persistentes criados em ODL, com a ajuda da OML
  - São mapeadas para C++, Java e Smalltalk

14

BDs Orientados a Objetos

- ODMG ODL C++
  - Possui tipos de dados persistentes
    - **d_Short**, **d_UShort** - 16 bits
    - **d_Long**, **d_ULong** - 32 bits
    - **d_Float** - 32 bits, formato IEEE
    - **d_Double** - 64 bits, formato IEEE
    - **d_Char**, **d_String** - 8 bit, formato ASCII
    - **d_Octet** - 8 bit, sem formato predefinido
    - **d_Boolean** – 1 bit, 0 (falso) ou 1 (verdadeiro)
  - Permite definir classes de objetos

```
Ex.: class Pessoa : public d_Object {
        public:  d_String Nome;
        private: d_Long CPF;
       };
```

15

BDs Orientados a Objetos

- ODMG ODL C++ (cont.)
  - Permite criar conjuntos de objetos

```
Ex.: class CorpoDocente : public d_Object {
        public:   d_Set<Pessoa> Professores;
       };
```

  - Podem existir relações de herança entre objetos

```
Ex.: class Aluno : public Pessoa {
        public:   d_Long Matrícula;
       };
```

  - Referências apontam para objetos

```
Ex.: class Turma {
        public:   d_String Disciplina;
                  d_Ref<Pessoa> Professor;
                  d_Set<d_Ref<Aluno>> Alunos;
       };
```

16

BDs Orientados a Objetos

- Exemplo de código em ODMG OML C++

```
void novo_aluno (d_String Nome, d_Long CPF,
  d_Long Matrícula) {
  d_Database db;
  d_Transaction trans;
  db.open("AlunosDB");
  trans.begin();
  d_Ref<Aluno> NovoAluno = new(db) Aluno;
  NovoAluno→Nome = Nome;
  NovoAluno→CPF = CPF;
  NovoAluno→Matrícula = Matrícula ;
  trans.commit();
  db.close();
}
```

17

BDs Orientados a Objetos

- Exemplo de código em ODMG OQL C++

```
void busca_prof (d_String nome) {
  d_Database db;
  db.open("CorpoDocenteDB");
  d_OQL_Query query("select Professores from
    CorpoDocente where Professores.Nome like :nome")
  d_Set<d_Ref<Pessoa>> profs;
  d_OQL_Execute(query, profs);
  d_Iterator<d_Ref<Pessoa>> iter =
    profs.create_iterator();
  while(iter.next(profs)) {
    cout << "Nome: " << profs.Nome
         << "   CPF: " << profs.CPF << endl;
  }
  db.close();
}
```

18

BDs XML

- XML é uma linguagem extensível de marcação de dados definida pelo W3C
- XML é usada para intercambiar dados
  - Permite trocar dados facilmente entre aplicações Web
  - Facilita a análise de dados por programas
  - É independente de sistemas operacionais ou formatos proprietários usados por aplicações
  - Permitindo a definição de elementos pelo usuário (ou aplicação) para estruturar dados

19

BDs XML

- Documentos XML
  - Documentos estruturados em formato texto
  - Compostos por tags XML e valores dos dados
  - Tags podem ser definidas pelo usuário
  - Legíveis para humanos e máquinas
  - Os dados contidos em um documento XML podem ser facilmente interpretados pelas aplicações, independentemente de linguagem de desenvolvimento, do sistema operacional e do protocolo de comunicação utilizado

20

BDs XML

- Documentos XML x HTML
  - XML é visto erroneamente como um formato alternativo ao HTML
  - XML não possui tags para formatação de documentos, como o HTML
  - XML se preocupa apenas com o conteúdo do documento, e não com a sua apresentação

21

BDs XML

- Apresentação de documentos XML
  - Os dados de um documento XML podem ser apresentados de várias maneiras, dependendo do contexto no qual são utilizados
  - Folhas de estilo XSL (*eXtensible Stylesheet Language*) especificam regras para apresentar um documento XML (em HTML, PDF, ...)
  - Diferentes folhas de estilo podem ser aplicadas a um mesmo documento XML, apresentando o dado de forma diferente em cada situação ou para diferentes usuários

22

BDs XML

- Elementos de um documento XML
  - Especificados usando *tags*
    - Tag de abertura: <tag>
    - Tag de fechamento: </tag>
    - Tag com auto-fechamento: <tag />
  - Os valores dos dados são especificados entre tags de abertura e fechamento:
    <tag>dado</tag>
  - Tags podem possuir atributos:
    <tag atrib="valor" />
  - Tags podem conter outras tags aninhadas:
    <tag1> <tag2>dado</tag2> </tag1>

23

BDs XML

- Documentos XML devem ser bem-formados
  - Devem conter apenas um elemento, que é a raiz da árvore XML
  - O elemento raiz pode conter outros elementos
  - Todos os elementos especificados em XML devem ser finalizados, ao contrário de HTML
  - Elementos aninhados devem ser finalizados na ordem inversa de abertura

24

BDs XML

- Exemplo de documento XML

```
<?xml version="1.0"?>
<clientes>
  <cliente id="1">
    <nome>João da Silva</nome>
    <cpf>123.456.789-00</cpf>
  </cliente>
  <cliente id="2">
    <nome>José dos Santos</nome>
    <cpf>987.654.321-00</cpf>
  </cliente>
</clientes>
```

25

BDs XML

- Esquemas XML
  - Especificam o formato que deve ser respeitado por um documento XML
  - Definem tags, atributos e os tipos de dados aceitos para cada elemento
  - Um documento XML é válido se estiver em conformidade com um esquema
  - Tipos de esquemas XML
    - DTD (*Document Type Definition*)
    - XSD (*XML Schema Definition*)

26

BDs XML

- DTD
  - Formato não-XML
  - Pode ser embutido no XML ou especificado em um arquivo em separado (extensão .dtd)
  - Especifica as tags aceitas, seus atributos e as tags que esta pode conter
  - Limitação: não define os tipos de dados e os valores aceitos em cada campo do documento

27

BDs XML

- Exemplo de DTD

```
<!ELEMENT clientes (cliente*)>
<!ELEMENT cliente (nome, cpf)>
<!ATTLIST cliente id CDATA #REQUIRED>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cpf (#PCDATA)>
```

28

BDs XML

- XSD
  - Formato XML
  - Permite especificar os tipos de dados, o formato e os valores aceitos em cada campo
  - Pode ser facilmente reutilizado em outros esquemas
  - Proposto pela Microsoft e posteriormente aceito como um padrão W3C

29

BDs XML

- Exemplo de XSD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="clientes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="cliente" type="xs:string"/>
      <xs:element name="cpf" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

30

BDs XML

- Parsers XML
  - São responsáveis por fazer a verificação de um documento XML, obtendo os dados que serão usados por uma determinada aplicação
- APIs XML
  - Usadas para fazer o parsing de documentos
  - Exemplos de APIs XML:
    - DOM
    - SAX

31

BDs XML

- XML pode ser usado em BDs como:
  - Formato para exportação e importação de dados: para troca de dados entre diferentes BDs relacionais
  - Formato de dados nativo do BD
- Bancos de Dados XML
  - Armazenam documentos XML
  - Empregam esquemas para definir e validar os dados
  - Usam linguagens de consulta próprias para acesso a dados em documentos XML
  - Fáceis de integrar com a Web e com outros sistemas

32

BDs XML

- XPath
  - *XML Path Language*
  - Linguagem para navegação na estrutura de um documento XML
- Características
  - Semelhante à navegação em diretórios
  - Permite consultar um documento XML sem conhecer o seu esquema
  - Busca por expressões regulares ou padrões
  - Permite o uso de predicados de seleção
  - Gera resultado (conjunto de elementos aninhados) em XML

33

BDs XML

- XPath – Expressões de caminho:

/	→ documento inteiro
/cliente/nome	→ todos os nomes de clientes
/*/cpf	→ todos os CPFs de elementos
/cliente//cpf	→ todos os CPFs de clientes
/cliente[1]	→ primeiro cliente
/cliente[last()]	→ último cliente
/cliente/cpf | /func/cpf	→ CPFs de clientes e funcionários
/cliente/@id	→ valores do atributo id
//cpf	→ qualquer elemento CPF

34

BDs XML

- XPath – Predicados:

/cliente[@id = "1"]	→ cliente com id = 1
/cliente[@id > 10]/cpf	→ CPF dos clientes com id > 10
/cliente/nome/[email]	→ nomes de clientes com e-mail
/cliente[nome = "João"]	→ cliente de nome João
/cliente[not(@id)]	→ clientes sem id
/cliente[count(filho) = 0]	→ clientes sem filhos
/cliente[nome contains(text(), "Silva")]	→ clientes com sobrenome Silva

35

BDs XML

- XPath – Limitações:
  - Recupera somente partes de um documento XML
  - É incapaz de produzir resultados mais elaborados nas consultas, como por exemplo:
    - Junção de dados em dois documentos XML
    - Criação de novos elementos XML
- XQuery
  - Nova proposta da W3C
  - Possui maior poder de expressão que a XPath

36

BDs XML

- XQuery – Consultas:

  for *variável* in *expressão XPath*
  [let *novas variáveis*]
  [where *condição*]
  return *resultado*

- XQuery – Exemplos:

```
for $c in /cliente
where $c/nome = "João da Silva"
return <cpf>
          { $c/cpf }
       </cpf>
```

37

BDs XML

- XQuery – Exemplos (cont.):

```
for $f in /func
let $renda := $f/salario + $f/bonus
where $f/@tipo = "sócio"
return {$f/nome, $renda}

for $c in /cliente
for $f in /func
where $c/@idfunc = $f/@idfunc
return {$f/nome, $c/nome}
sort by (. ascending)
```

38