



Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA
Departamento de Informática
Integrado / Análise e Desenvolvimento de Sistemas / Licenciatura em Computação

Conceitos Básicos de Banco de Dados - PARTE 2

André L. R. Madureira <andre.madureira@ifba.edu.br>
Doutorando em Ciência da Computação (UFBA)
Mestre em Ciência da Computação (UFBA)
Engenheiro da Computação (UFBA)

Modelos de Dados

- Formas de descrever os **esquemas físico, lógico e de view** de um DB
 - Modelo relacional
 - Modelo entidade/relacionamento
 - Modelo baseado em objetos
 - Modelo semi-estruturado

Modelo relacional

Atributo ou
campo

Nome	Salário	Idade
Ana	1500	29
Jose	1212	21
Claudio	2500	32

Registro
ou Tupla

Valor do
atributo

Modelo relacional:

Usa tabelas para representar os dados e as relações entre eles

Cada linha é uma **instância**
Instância = **registro** = **tupla**

Cada coluna é um **atributo**
(campo)

Modelo relacional

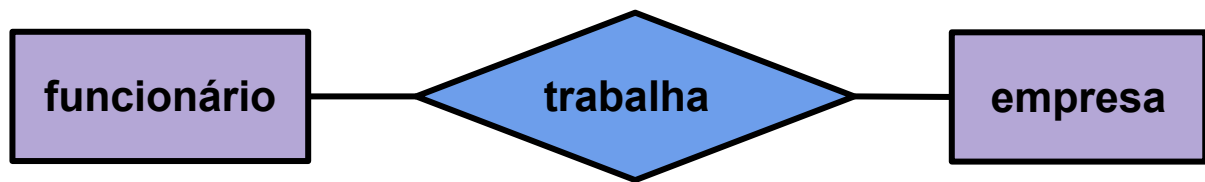
Coluna (atributo)

Paciente	CPF	Tipo Sanguineo
Marcela	111111	A+
João	22222	B-
Pablo	452452	AB+

Linha (registro)

Valor do Atributo

Modelo entidade/relacionamento (E-R)

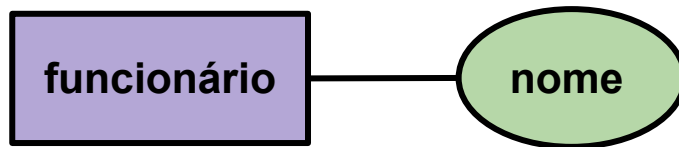


Entidade

Relacionamento

Entidade

Um funcionário
trabalha em uma
empresa



Entidade

Atributo

Um funcionário
possui um nome

Entidade = objetos
do mundo real

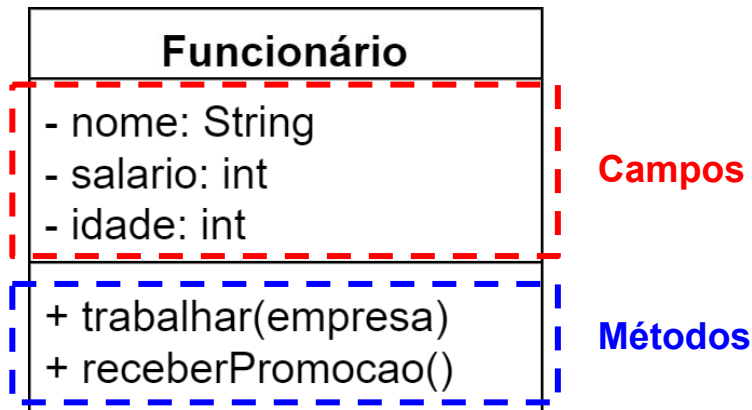
Relacionamento =
associação entre entidades

Atributo = propriedade que
uma entidade possui

Modelo baseado em Objetos

Classe:

Define quais dados podem ser armazenados (**campos**) e como manipulá-los (**métodos**)



Objeto:

É uma instância de uma classe

funcionario1 : Funcionário

nome = " Ana "
salario = 1500
idade = 29

funcionario2 : Funcionário

nome = "Jose"
salario = 1212
idade = 21

Modelo baseado em Objetos

Modelo baseado em objetos:

Dados são representados por objetos

Objeto “funcionario1”

funcionario1 : Funcionário
nome = " Ana "
salario = 1500
idade = 29

Objeto “funcionario2”

funcionario2 : Funcionário
nome = "Jose"
salario = 1212
idade = 21

Instâncias = objetos

Classe “Funcionario”

Funcionário
- nome: String - salario: int - idade: int
+ trabalhar(empresa) + receberPromocao()

Esquema = conjunto de classes

Modelo semi-estruturado

Modelo

semi-estruturado:

Possui instâncias com
formato variável

Instâncias com formato variável:

As instâncias podem
possuir diferentes
conjuntos de atributos

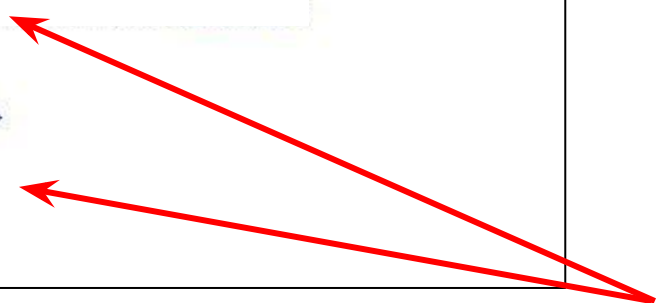
```
{  
  "Funcionario": {  
    "001": {  
      "nome": "Ana",  
      "salario": 1500,  
      "idade": 29  
    },  
    "002": {  
      "nome": "Jose",  
      "salario": 1212  
    },  
    "003": {  
      "nome": "Claudio",  
      "salario": 2500,  
      "idade": 32,  
      "formacao": "Engenheiro"  
    }  
  }  
}
```

Está faltando a idade
de **José**!

Cláudio é o único que
possui o atributo
“**formação**”

Modelo semi-estruturado

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <MUSICAS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <MUSICA>
  <NOME>A Fórmula Do Amor</NOME>
  <CANTOR>Kid Abelha</CANTOR>
  <LETRA>Eu tenho gestos aptos</LETRA>
</MUSICA>
- <MUSICA>
  <NOME>A Viagem</NOME>
  <CANTOR>Roupa Nova</CANTOR>
</MUSICA>
- <MUSICA>
  <NOME>Águas De Março</NOME>
  <CANTOR>Elis Regina</CANTOR>
</MUSICA>
</MUSICAS>
```

The diagram consists of two red arrows originating from the right edge of the slide. One arrow points to the second XML element, which is missing the 'LETRA' attribute. The other arrow points to the third XML element, which is also missing the 'LETRA' attribute. These arrows highlight the semi-structured nature of the XML where certain attributes are not consistently present.

Exemplo: Arquivo XML com instâncias do tipo “**MUSICA**” faltando o atributo “**LETRA**”

Modelo semi-estruturado

- **Por que usar esse modelo?**
 - Flexibilidade na definição dos atributos
 - Não é preciso projetar o esquema antecipadamente
 - Alto desempenho
 - DBMS não precisa verificar se todos os atributos estão presentes em cada registro
- Modelo mais usado para aplicações de **Big Data**

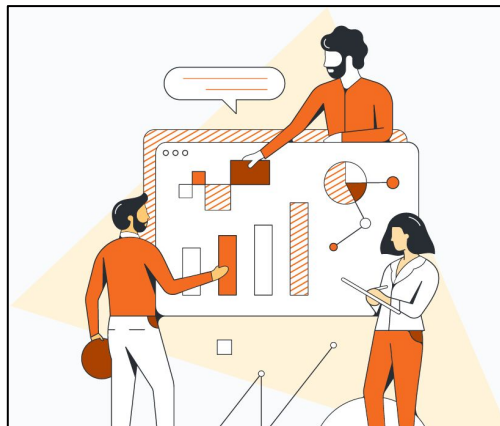


Atividade - Construir modelos para Padaria

- Construir modelos de dados entidade-relacionamento e relacional para uma padaria que possui os seguintes requisitos:
 - Controle de estoque (id, produto, qtd)
 - Controle de vendas de produtos (funcionário, data_e_horario, cliente, produto_vendido, qtd_vendida, preco_venda)
 - Relatório de total de vendas mensal
 - Quais produtos venderam mais?
 - Quais deram mais lucro para a padaria?

Linguagens de Banco de Dados

- Bancos de dados utilizam duas linguagens:
 - Linguagem de **M**anipulação de Dados (**DML**)
 - Linguagem de **D**efinição de Dados (**DDL**)



Linguagem de Manipulação de Dados (DML)

- Permite a manipulação de dados pelos usuários:
 - Inserção (**insert**)
 - Atualização (**update**)
 - Remoção (**delete**)
 - Consulta (**select**)



Linguagem de Manipulação de Dados (DML)

Tabela funcionários

Nome	Salário	Idade
Ana	1500	29
Jose	1212	21
Claudio	2500	32

**Inserir os
dados na
tabela**



```
INSERT INTO funcionarios VALUES (  
    1,  
    'Ana',  
    1500,  
    29  
);  
INSERT INTO funcionarios VALUES (  
    2,  
    'Jose',  
    1212,  
    21  
);  
INSERT INTO funcionarios VALUES (  
    3,  
    'Claudio',  
    2500,  
    32  
);
```

Linguagens de Manipulação de Dados (DML)

- **Procedural**

Descreve “**como**” obter os dados que desejamos manipular

```
create PROCEDURE simpleprocedure (inval NUMBER)
IS
  tmpvar    NUMBER;
  tmpvar2   NUMBER;
BEGIN
  tmpvar := 0;
  tmpvar2 := 0;
  FOR lcv IN 1 .. inval
  LOOP
```

- **Declarativas**

Descreve quais dados devem ser obtidos,
MAS NÃO COMO fazer isso

```
SELECT nome, salario, idade FROM funcionario;
SELECT cia, horario, preco FROM passagens_aviao;
```

Linguagens de Manipulação de Dados (DML)

- **Procedural**

- Usuário especifica os dados que deseja obter e **“como”** obtê-los
- O usuário controla o desempenho do DB ao realizar uma consulta

- **Declarativas**

- Usuário especifica os dados que deseja obter porém não define **“como”**
- Mais fácil de usar

Linguagens de Definição de Dados (DDL)

- Especifica os esquemas do banco de dados

Tabela funcionários

Nome	Salário	Idade
Ana	1500	29
Jose	1212	21
Claudio	2500	32

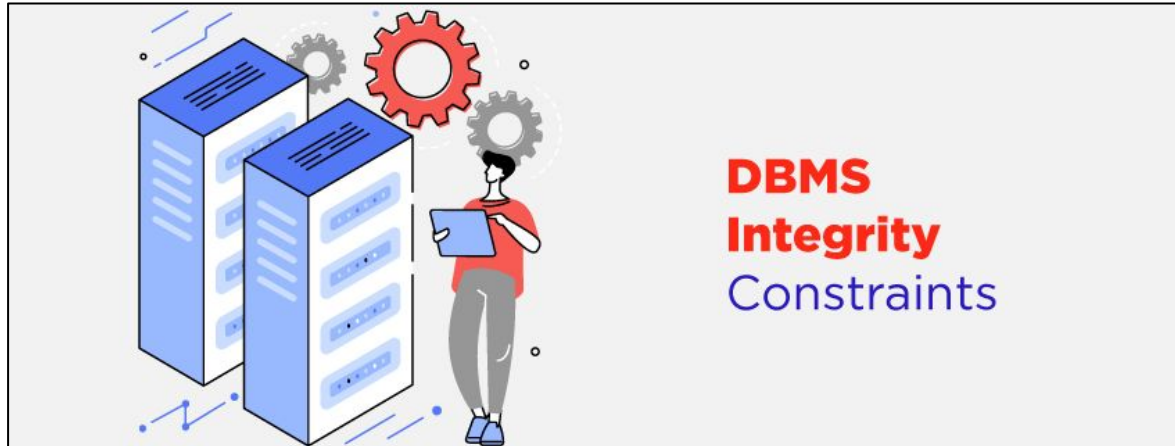
**Criação
da tabela**



```
CREATE TABLE funcionarios (  
    id INTEGER PRIMARY KEY,  
    nome VARCHAR(250),  
    salario INTEGER,  
    idade INTEGER  
);
```

Linguagens de Definição de Dados (DDL)

- Permitem definir **restrições de integridade**:
 - São regras que são verificadas pelo banco de dados sempre que ele sofre uma atualização



Classificação de Restrições de Integridade

- **Restrições de domínio e de entidade**
- **Restrições de chave**
- **Integridade Referencial**
- **Assertivas**
- **Autorização** (leitura, inserção, atualização, exclusão)

Restrições de domínio e de entidade

Restrições de Domínio e de Entidade:

Definem o conjunto de valores que um atributo pode assumir.

Ex: inteiro, caractere, valores nulos, etc

```
mysql> DESC Employee_details;
```

Field	Type	Null	Key	Default	Extra
EmployeeID	int	NO	PK1	NULL	
Name	varchar(255)	NO		NULL	
Address	varchar(255)	YES		NULL	
Salary	decimal(18,2)	YES		NULL	

4 rows in set (8.05 sec)

- O atributo **"Name"**:
 - Deve conter um texto com até 255 caracteres
 - Não pode ter valores vazios (**nulos**)
 - Isto é, todo funcionário deve ter um nome
 - **Ex:** "Jeferson"

Restrições de chave

Restrições de Chave:

Definem se o mesmo valor de um atributo pode se repetir em outro registro do banco de dados.

Ex: registros com valores duplicados

```
mysql> DESC Employee_details;
```

Field	Type	Null	Key	Default	Extra
EmployeeID	int	NO	PRI	NULL	
Name	varchar(255)	NO		NULL	
Address	varchar(255)	YES		NULL	
Salary	decimal(18,2)	YES		NULL	

4 rows in set (8.05 sec)

- O atributo “**EmployeeID**” tem uma restrição de chave primária
 - Registros diferentes devem ter valores diferentes de “**EmployeeID**”
 - **Ex:** há somente um funcionário com id 1, e ninguém mais
 - Outros funcionários devem ter outros ids

Integridade Referencial

Tabela funcionario

ID	Nome	Salário	Idade	idChefe
1	Ana	1500	29	1002
2	Jose	1212	21	1000
3	Claudio	2500	32	1001

Tabela chefe

ID	Chefe	Cargo
1000	Marcia	Gerente geral
1001	Ana	Gerente regional
1002	Romario	Sócio

Integridade Referencial

Cada funcionário da tabela **“funcionario”** possui um chefe, identificado por **“idChefe”**

Deve existir um chefe na tabela **“chefe”** com o **“idChefe”** de cada funcionário

Exemplo de integridade referencial

Funcionario				
id	nome	salario	idade	id_chefe
1	Ana	1500	29	1002
2	Jose	1212	21	1002
3	Claudio	2500	32	1001

Chefe		
id	nome	cargo
1000	Marcia	Gerente geral
1001	Ana	Gerente regional
1002	Romario	Socio

Podemos ter vários funcionários associados ao mesmo chefe

Seja lá qual for o “*id_chefe*” do funcionário, ele SEMPRE deve existir na tabela “*Chefe*”

Exemplo de falha na integridade referencial

Funcionario					Chefe		
id	nome	salario	idade	id_chefe	id	nome	cargo
1	Ana	1500	29	1002	1000	Marcia	Gerente geral
2	Jose	1212	21	1002	1001	Ana	Gerente regional
3	Claudio	2500	32	99 ??	1002	Romario	Socio

Pelo menos um dos “*id_chefe*”
de funcionários não existe na
tabela “*Chefe*”



O DBMS vai dar erro. Isto é, o
DBMS se recusa a inserir ou
modificar o registro com “*id_chefe*”
incorreto

Assertivas (*Assertions*)

- Condições que o DB sempre verifica e garante que permaneçam verdadeiras
 - **Ex:** deve existir ao menos um funcionário na empresa

```
CREATE ASSERTION funcionarioCheck CHECK (  
    NOT EXISTS ( SELECT * FROM funcionarios )  
);
```

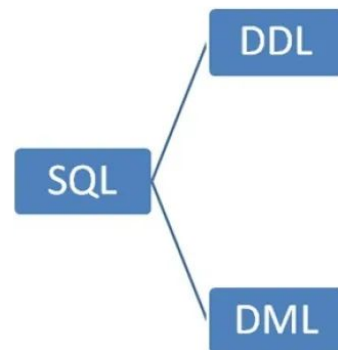
Linguagens de Definição de Dados (DDL)

- **Autorização e Controle de Acesso**
 - Define o usuário que pode **inserir, ler, alterar** ou **deletar** dados de quais tabelas
 - **Ex:** permitir que o usuário **ANA** altere dados da tabela “**tabela_funcionarios**”

```
GRANT UPDATE ON tabela_funcionarios TO ANA WITH GRANT OPTION;
```

DDL e DML na prática

- Na prática, as linguagens DDL e DML são parte de uma mesma linguagem
 - **Ex:** Linguagem SQL
 - Linguagem mais utilizada em SGBDs
 - **É declarativa (não procedural)**
 - Descrevemos os dados que queremos obter através de **consultas SQL**



DDL e DML na prática

- Como o SGBD executa uma consulta SQL?
 - Há um módulo de **processamento e otimização de consulta** no SGBD
 - O módulo escolhe um **plano de execução** eficiente para cada consulta
 - **Plano de execução:** conjunto de operações para buscar as informações no banco de dados

Otimização de plano de execução

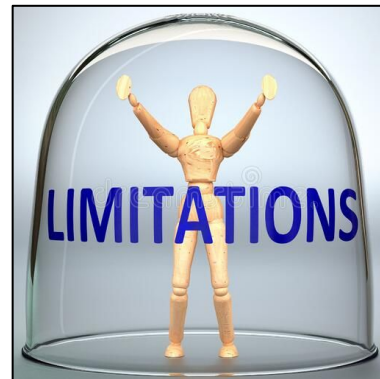
- É possível otimizar planos de execução através de **índices** criados no SGBD
 - **Índice:** Arquivos auxiliares que aceleram a pesquisa de dados em disco (seja ele um HDD, SSD, ou outro)
 - **Consequência:** os índices consomem espaço em disco

Índices e o desempenho de SGBDs

- Índices são modificadas quando os dados a quem eles estão associados são alterados no DB
 - **Ex:** Um índice para o atributo “endereço” de uma pessoa é atualizado sempre que o endereço da pessoa mudar
- Se um SGBD possuir muitos índices que mudam com frequência teremos **perdas de desempenho**

Limitações da Linguagem SQL

- SQL permite definir, manipular e realizar operações sobre os dados
- Porque não podemos usar somente o SQL para programar?
 - A linguagem SQL **NÃO é Turing completa**
 - Existem operações que o SQL não consegue realizar

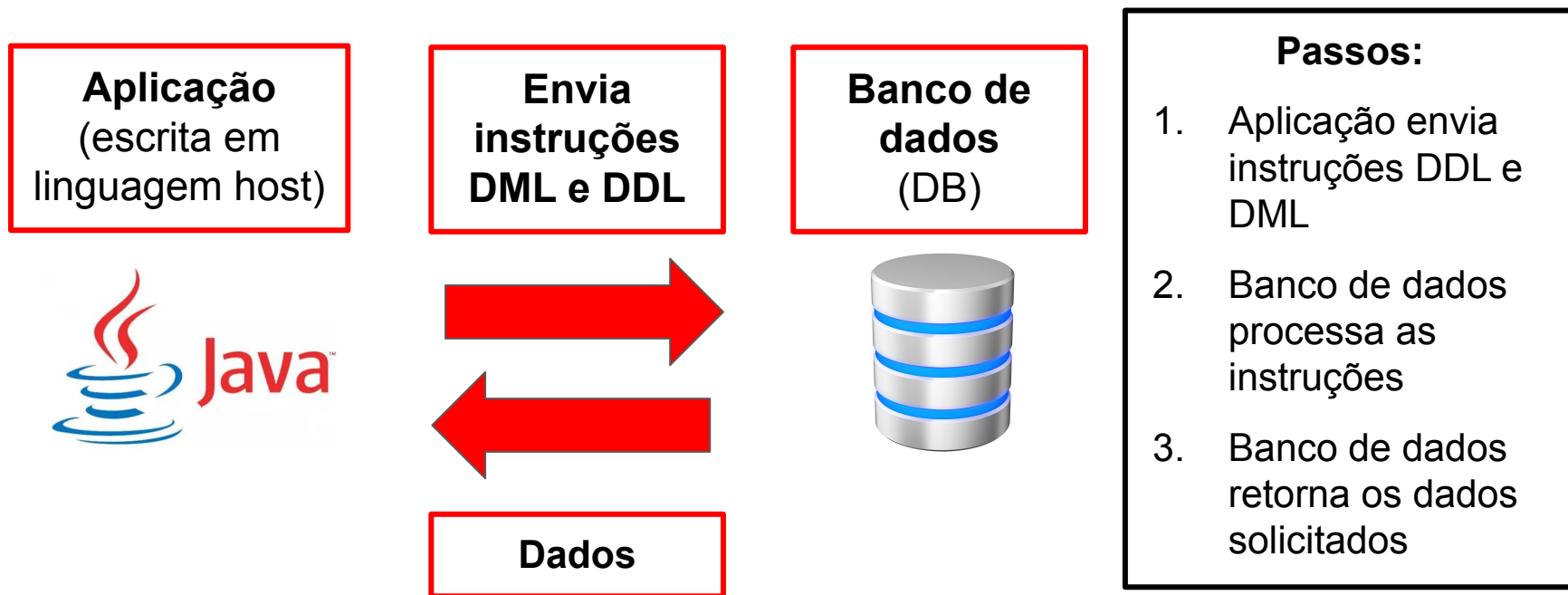


Limitações da Linguagem SQL

- A linguagem SQL **NÃO é Turing completa**
 - Existem operações que o SQL não consegue realizar
- **Solução:**
 - Precisamos de **linguagens de host** para auxiliar o SQL
 - **Linguagens host** = linguagens programação tradicionais
 - **Ex:** C, C++, Java

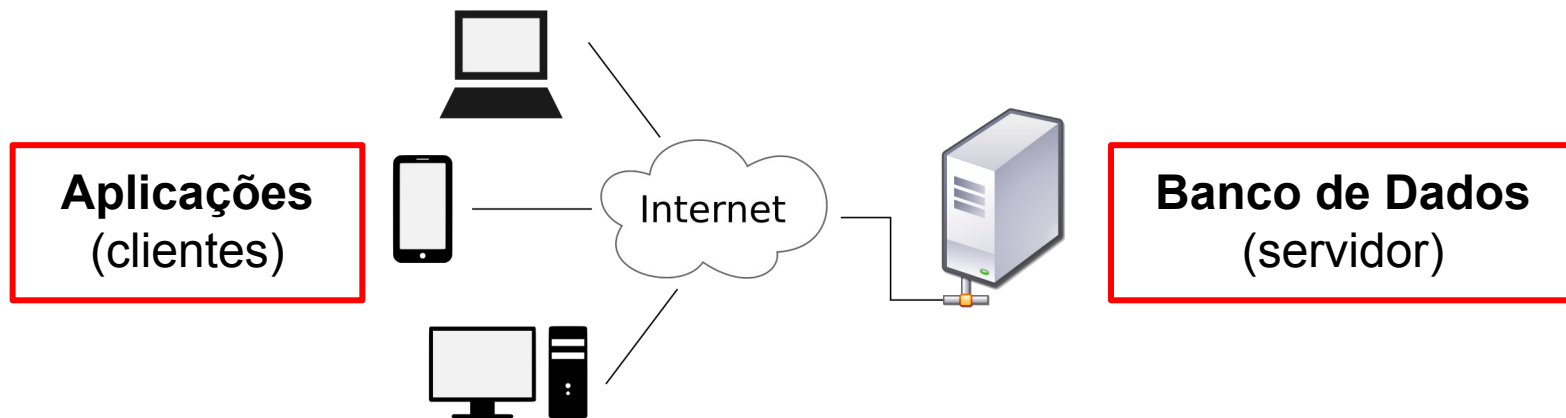


Interação entre Linguagem Host e DB



Arquitetura de Duas Camadas

- As aplicações e os bancos de dados são executadas em máquinas distintas
- Aplicações enviam instruções para o banco de dados



Arquitetura de Duas Camadas

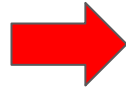
- Sistemas complexos possuem **lógica empresarial**
 - **Lógica Empresarial** = ações que devem ser executadas de acordo com um conjunto de condições
 - **Ex:** só transfira dinheiro de uma conta para outra se houver saldo para isso



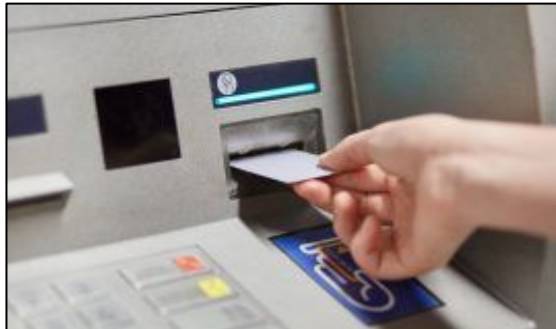
Problemas da Arquitetura de Duas Camadas

- Lógica Empresarial dentro da aplicação do cliente
 - Vulnerabilidades de segurança

Falha na aplicação do
caixa eletrônico

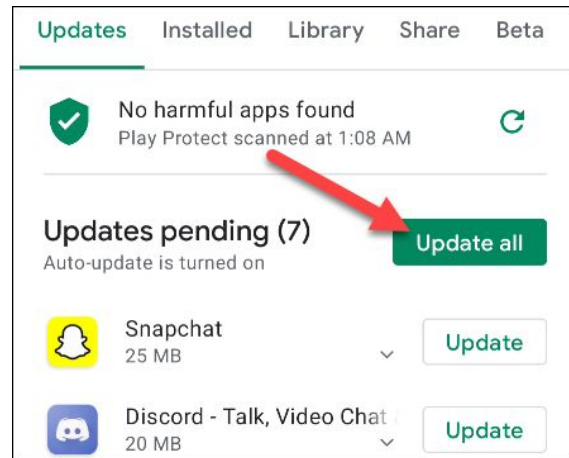


Pessoa SAQUE dinheiro SEM
reduzir o SALDO da sua conta

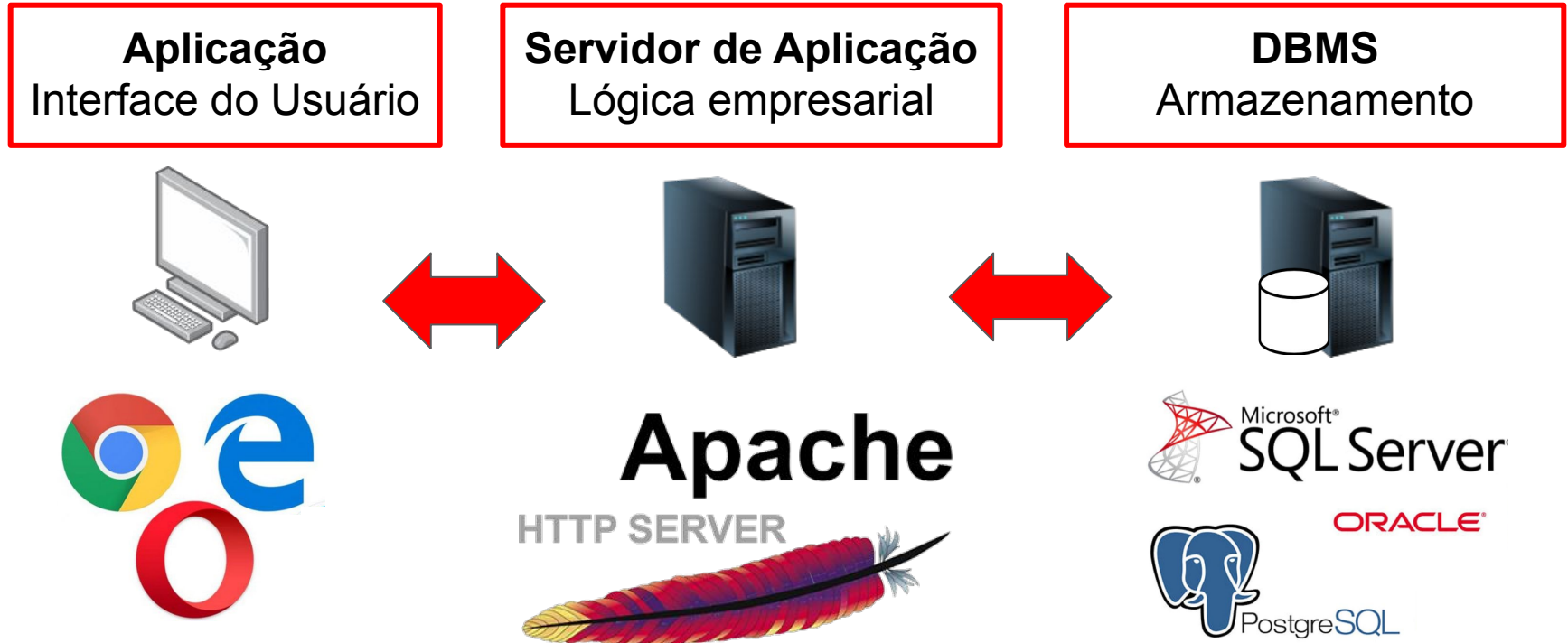


Arquitetura de Duas Camadas - Problemas

- Lógica Empresarial dentro da aplicação do cliente
 - Mudanças na lógica empresarial exigem alterações na aplicação do cliente
 - Como garantir que todos os clientes serão atualizados ao mesmo tempo?



Arquitetura de Três Camadas



Arquitetura de Três Camadas

- A aplicação NÃO INTERAGE diretamente com o DBMS!
 - Aplicação => servidor de aplicação => DBMS
 - Lógica empresarial está no servidor de aplicação



Referencial Bibliográfico

- KORTH, H.; SILBERSCHATZ, A.; SUDARSHAN, S. **Sistemas de bancos de dados**. 5. ed. Rio de Janeiro: Ed. Campus, 2006.
- DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Ed. Campus, 2004. Tradução da 8ª edição americana.