Programação II

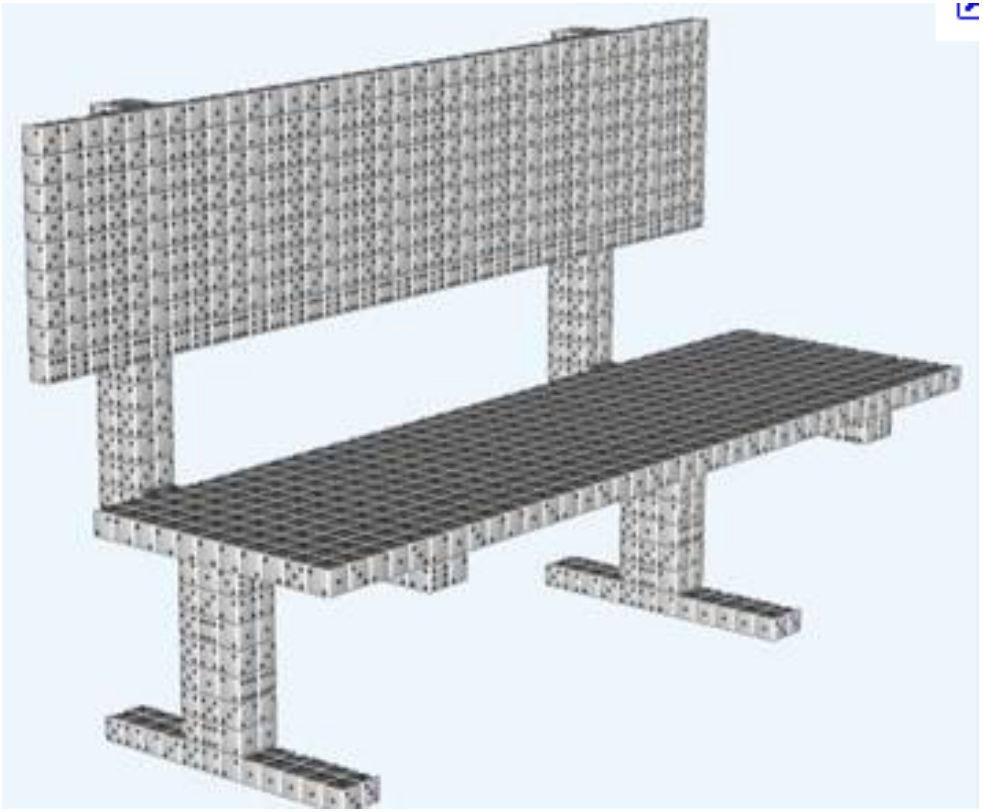
Introdução à Banco de Dados

http://dl.dropbox.com/u/3025380/prog2/aula8.pdf

flavio.ceci@unisul.br

Introdução

- O que é um banco de dados?



Introdução

- Banco de dados pode ser definido como um conjunto de "**dados**" devidamente **relacionados**;

- Por "dados" podemos compreender como "**fatos conhecidos**" que podem ser **armazenados** e que possuem um **significado implícito**.

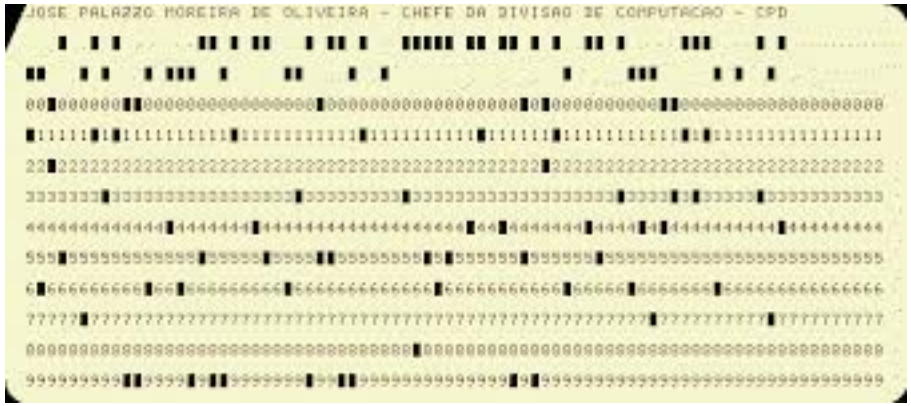
Introdução

- Um banco de dados possui as seguintes propriedades:
  - É uma **coleção lógica coerente** de dados com um **significado inerente**;
  - Uma disposição **desordenada** dos dados **não pode** ser referenciada como um banco de dados;

História dos Sistemas de BD

História dos Sistemas de BD

- **Década de 1950 e início da década de 1960:**
  - Processamento de dados usando fitas magnéticas para armazenamento:
    - Fitas fornecem apenas acesso sequencial;
  - Cartões perfurados para entrada.



História dos Sistemas de BD

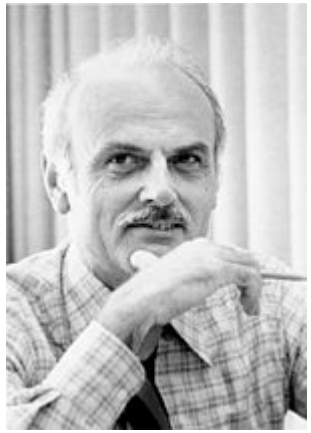
- **Final da década de 1960 e década e de 1970**:
  - Discos rígidos permitem acesso direto aos dados;
  - Ted Codd define o modelo de dados relacional:
    - Ganharia o **ACM Turing Award** por este trabalho;
    - IBM Reserach inicia o protótipo do System.
    - UC Berkeley inicia o protótipo do Ingres.

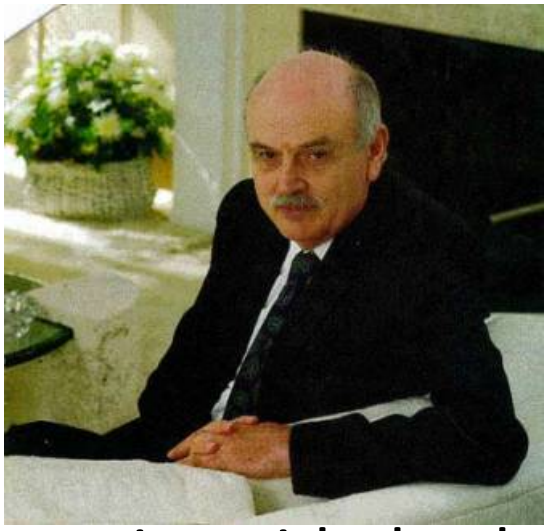
http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

  - Processamento de transação de alto desempenho (para a época).

História dos Sistemas de BD

- Edgar Frank “Ted” Codd: *(1923 - 2003)*





  - Recebeu seu PhD em 1965 pela universidade de Michigan;
  - Inventou o modelo relacional enquanto trabalhava na IBM.

História dos Sistemas de BD

- **Década de 1980**:
  - Protótipo relacionais de pesquisa evoluem para sistemas comerciais;
    - SQL se torna o padrão do setor;
  - Sistemas de banco de dados paralelos e distribuídos;
  - Sistemas de banco de dados orientados a objeto.

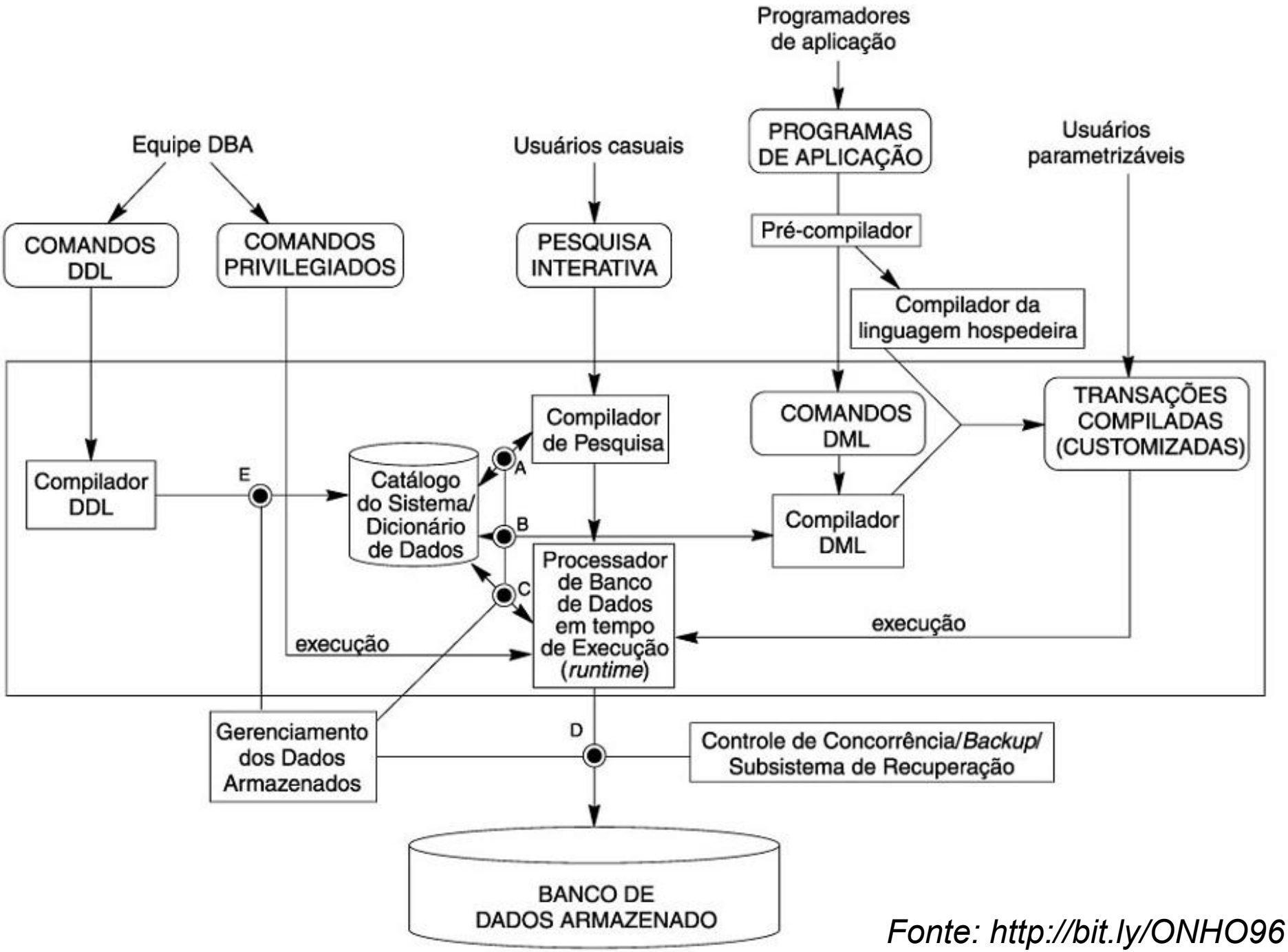
História dos Sistemas de BD

- **Década de 1990**:
  - Grandes aplicações de suporte a decisão e exploração de dados;
  - Grandes data warehouses de vários terabytes;
  - Surgimento do comércio Web.

História dos Sistemas de BD

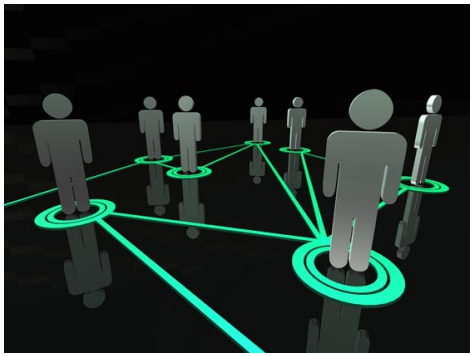
- **Década de 2000**:
  - Padrões XML e Xquery;
  - Administração de banco de dados automatizados.

Componentes de um SGBD



Fonte: http://bit.ly/ONHO96

Entidade e Relacionamento



Entidade

- São classes, cujos objetos representam **elementos existentes no mundo real** (nomes, pessoas, valores, documentos, etc.), em torno dos quais o software é desenvolvido.

- Também são conhecidas como classes de domínio da aplicação.

Entidades

- Um objeto pode ser considerado persistente quando seus atributos podem ser armazenados ao término de uma sessão de utilização do software e recuperado ao início de uma outra sessão.

- Existem diversas alternativas para persistir objetos
  - **Banco de dados relacional**;
  - Banco de dados orientado a objetos;
  - Serialização de objetos em sistemas de arquivos;
  - Documentos XML.

Entidades

- Diante do repositório de dados, um objeto estará sujeito a quatro operações de persistência (CRUD):
  - Criação, inserção ou inclusão no repositório (Create);
  - Leitura ou recuperação dos dados (Read);
  - Atualização ou modificação dos dados (Update);
  - Remoção ou eliminação dos dados (Delete).

Relacionamentos

- No desenvolvimento de software de complexidade média ou elevada, os modelos de dados recorrem aos **relacionamentos entre entidades** para expressar **situações do mundo real**;

- A UML define **associações** como sendo o **relacionamento entre os objetos**, em tempo de execução.

Relacionamentos

- Nos modelos de dados relacionais as associações de objetos encontram suporte nas **chaves estrangeiras** estabelecidas entre tabelas;

- A cardinalidade está diretamente ligada ao tipo de relacionamento entre as entidades.

Composição ***Um-para-Um***

Composição Um-para-Um

- Em determinadas situações a granularidade do modelo de classes pode ser maior do que a granularidade do modelo de banco de dados.

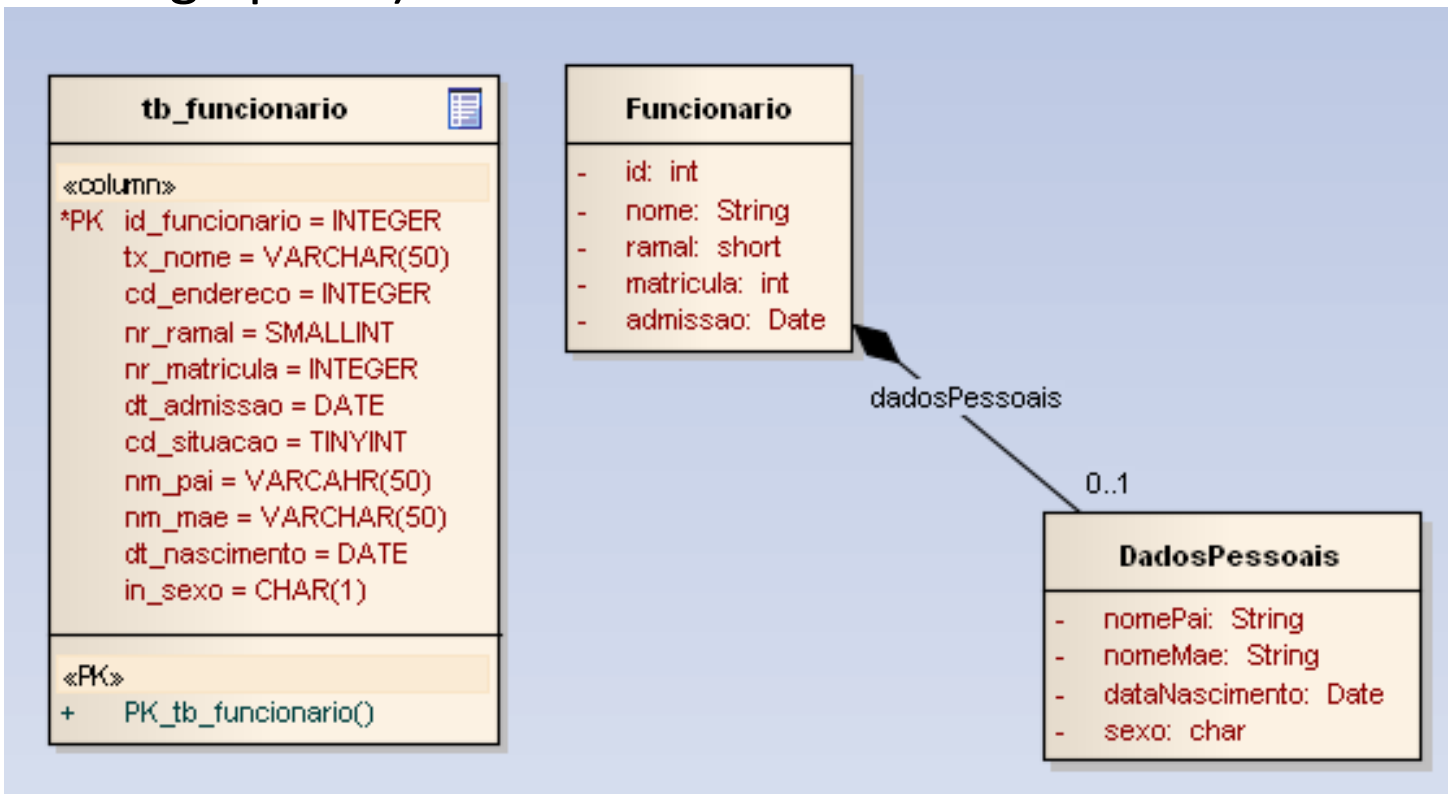
  – Por exemplo, duas classes que se relacionam na forma **um para um** podem ser mapeadas para uma única tabela no banco de dados.

Composição Um-para-Um

- Exemplo:
  - Vamos considerar que cada funcionário possui um conjunto de informações modeladas como seus dados pessoais.
  - Tal conjunto de informação existe em função de um funcionário, e é intransferível. Isto caracteriza um relacionamento de composição, com cardinalidade um para um.

Composição Um-para-Um

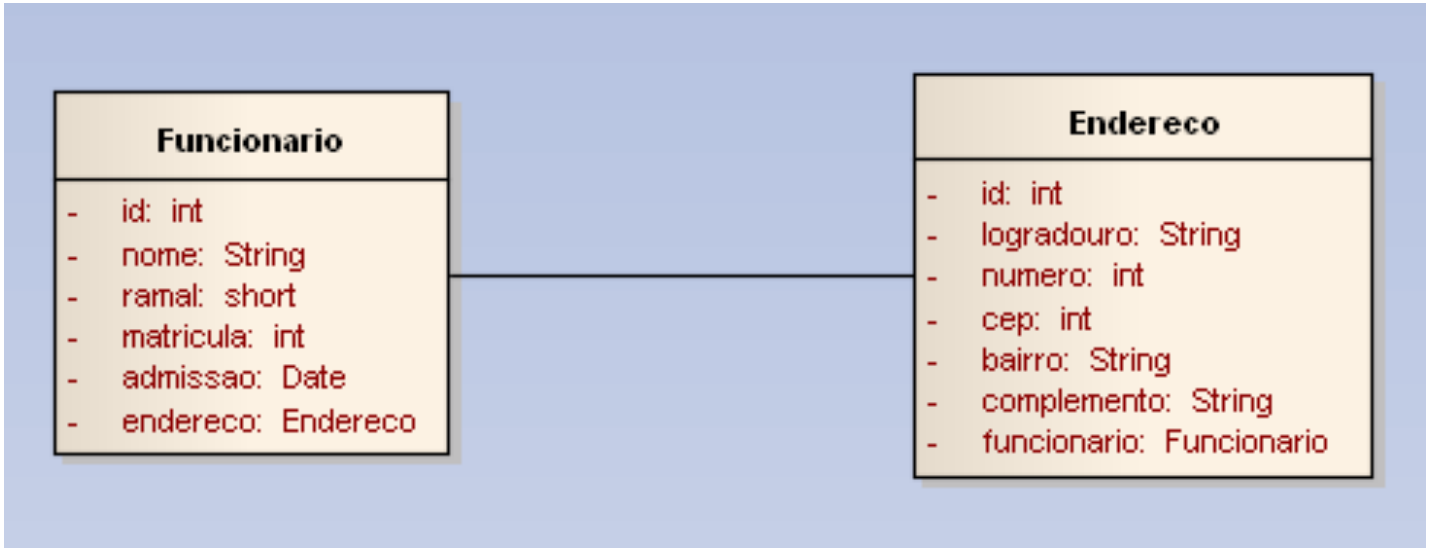
– A composição em orientação a objetos define uma associação forte, onde as partes estarão sempre associadas ao mesmo objeto todo (classe marcada com o losango preto).



Associação ***Um para Um***

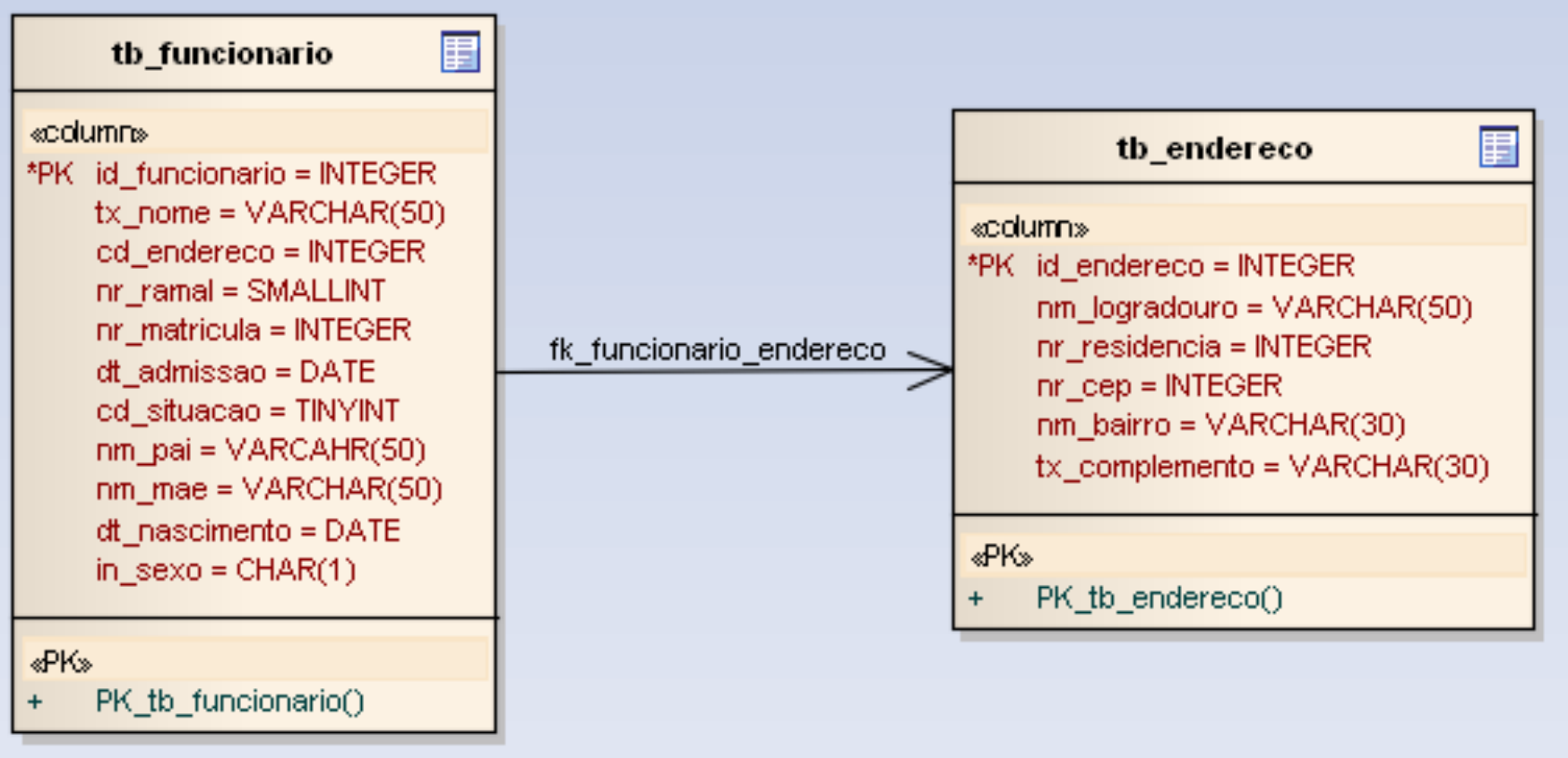
Associação Um-para-Um

- Mapeamento Um-para-Um e Navegação Bidirecional:
  - O primeiro caso de associação a ser estudado é a associação bidirecional entre Funcionario e Endereco:



Associação Um-para-Um

- Nos bancos de dados relacionais, é possível estabelecer o seguinte modelo de tabelas:



Associação Um-para-Um

- A coluna cd_endereco de tb_funcionario é a coluna que suporta o relacionamento.

- Consideramos então os funcionários como os “proprietários” do relacionamento.

- Observe que, na coluna cd_endereco, há uma chave estrangeira (***foreign key***) está estabelecendo o caminho unidirecional “do funcionario para o endereço”.

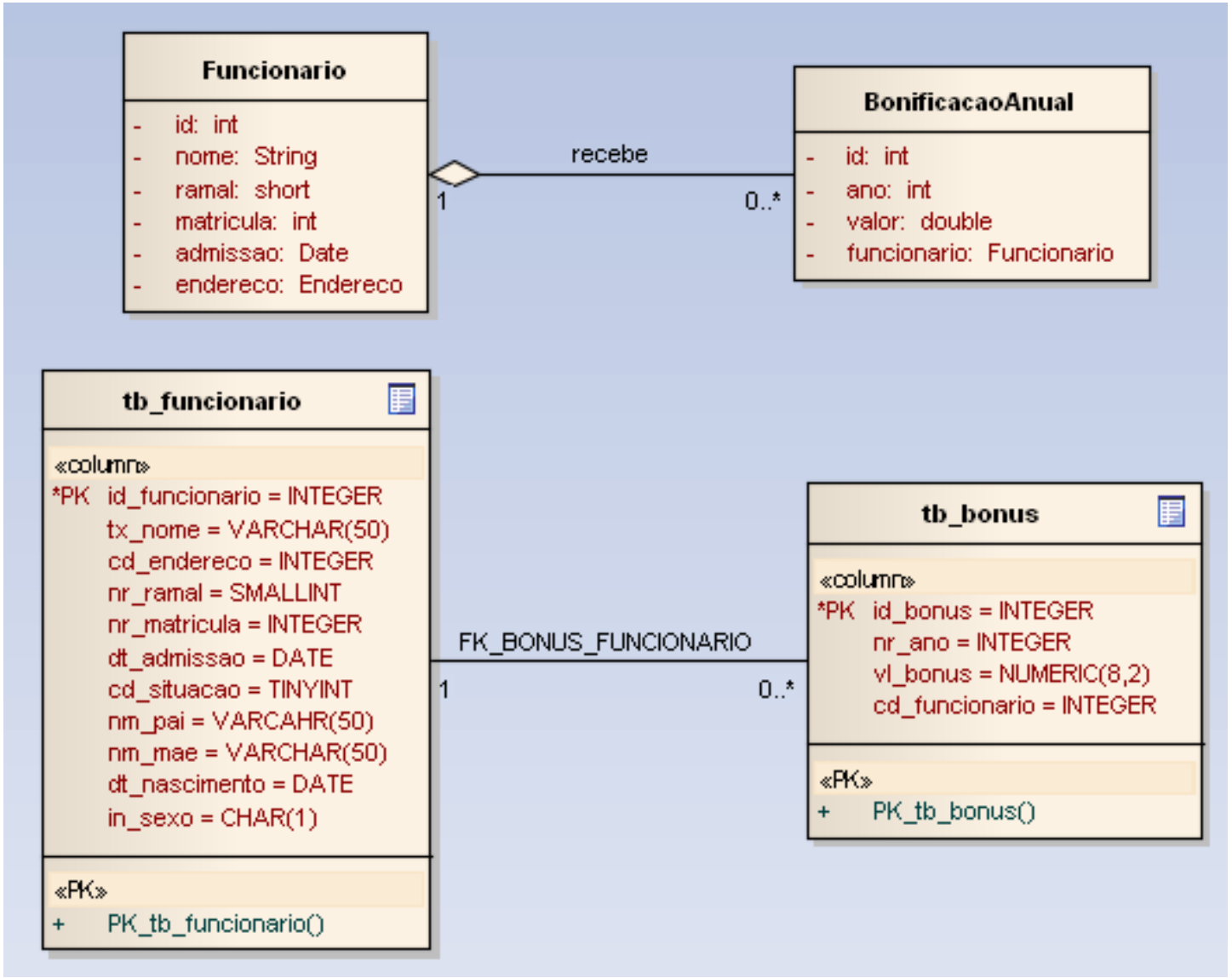
Associação ***Um para Muitos***

Associação Um-para-Muitos

- Freqüentemente os modelos de dados apresentam relacionamentos do tipo “um-para-muitos”.

- Será considerado o caso “Um funcionário recebe muitas bonificações”;
- Inicialmente vamos considerar a navegação bidirecional.

Associação Um-para-Muitos



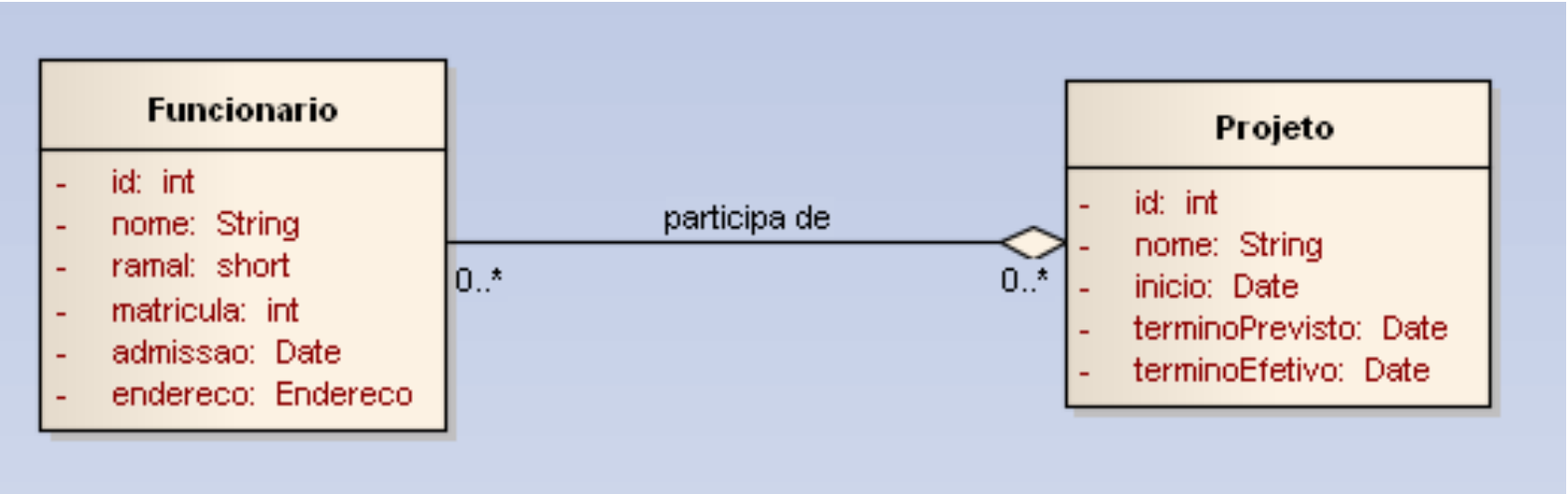
Associação Um-para-Muitos

- No banco de dados é necessário uma coluna de que suporta o relaciona na tabela “muitos”. Neste caso, trata-se da coluna cd_funcionario da tabela tb_bonus. É desejável a presença de uma chave estrangeira nesta coluna.
- ***Em Java existe várias formas de implementar este tipo de associação, com o auxilio de conjuntos, listas e mapas;***

Associação ***Muitos-para-Muitos***

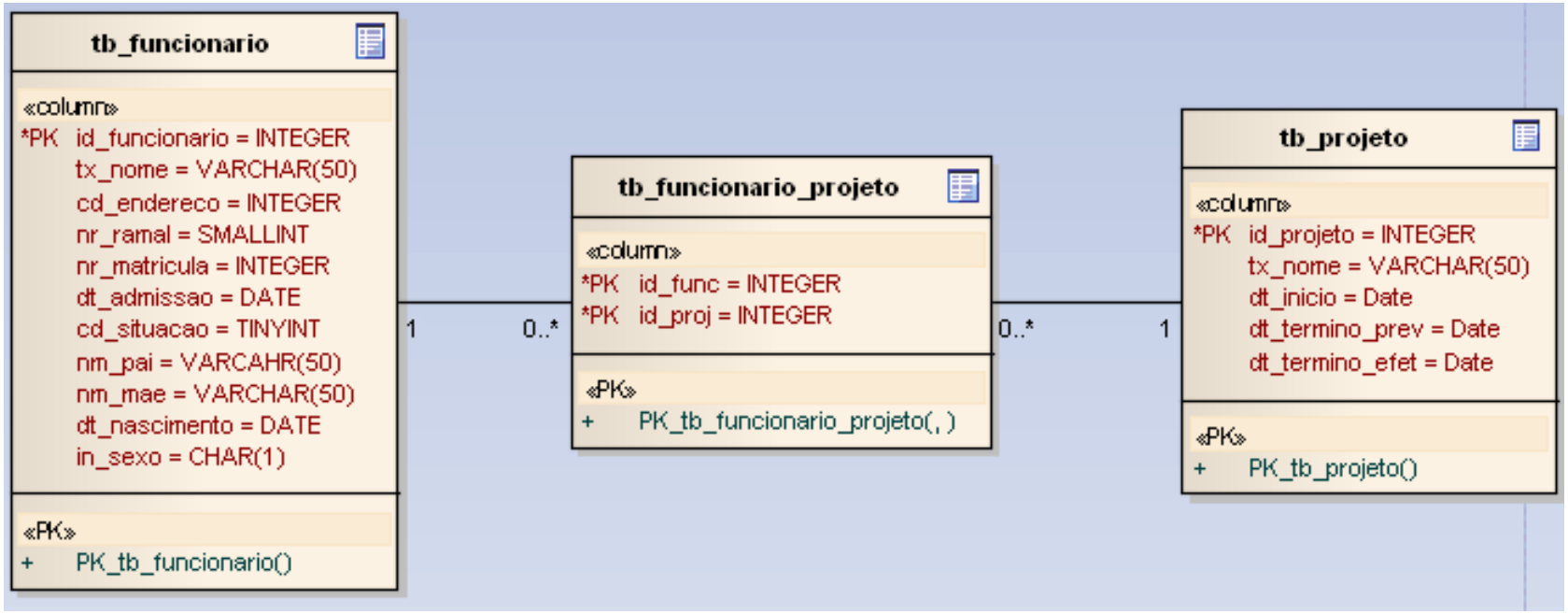
Associação Muitos-para-Muitos

- É o caso onde as duas extremidades de uma associação têm multiplicidade “muitos”.



Associação Muitos-para-Muitos

- No modelo relacional surge uma tabela intermediária para associar muitos funcionários a muitos projetos:




SQL

SQL

- Tipos de dados:

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to 9223372036854775807
decimal	variable	user-specified precision, exact	no limit
numeric	variable	user-specified precision, exact	no limit
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
serial	4 bytes	autoincrementing integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Criação e remoção de tabelas

SQL-DDL (Criando tabelas)

- Defini-se uma relação SQL usando o comando: ***CREATE TABLE.***

CREATE TABLE nome_tabela (A1 D1, A2 D2, <regra de integridade>);

  - cada *Ai* é o nome de um atributo no esquema da tabela;
  - *Di* é o tipo de domínio (tipo do dado) dos valores do atributo;

SQL-DDL (Criando tabelas)

- As principais regras de integridade são:

  primary key(Aj1, Aj2, ..., Ajm):

  – A especificação primary key diz que os atributos: Aj1, Aj2, ..., Ajm formam a chave primária da relação (tabela);

  – É necessário que a chave primaria seja não nula e única!

SQL-DDL (Criando tabelas)

- As principais regras de integridade são:

  foreign key(Aj1, Aj2, ..., Ajm) references r:

  – A cláusula foreign key inclui a relação dos atributos que constituem a chave estrangeira (Aj1, Aj2, ..., Ajm);

  – É utilizando ***r*** quanto ao nome da relação à qual a chave estrangeira faz referencia.

SQL-DDL (Criando tabelas)

- EXEMPLO:

```
CREATE TABLE TESTE (
    COD_TESTE INTEGER PRIMARY KEY,
    NOME_TESTE VARCHAR
);


CREATE TABLE TESTE2 (
    COD_TESTE2 INTEGER PRIMARY KEY,
    COD_TESTE INTEGER REFERENCES TESTE,
    NOME_TESTE2 VARCHAR
);

CREATE TABLE TESTE3(
    COD_TESTE INTEGER,
    COD_TESTE2 INTEGER,
    NOME_TESTE3 VARCHAR,
    FOREIGN KEY (COD_TESTE) REFERENCES TESTE,
    FOREIGN KEY (COD_TESTE2) REFERENCES TESTE2,
    PRIMARY KEY (COD_TESTE, COD_TESTE2)
);
```

SQL-DDL (Remover Tabelas)

- O comando ***DROP TABLE*** remove todas as informações de uma relação do banco de dados.
- Tanto os dados como o esquema é excluído.

```
DROP TABLE <nome_tabela>;
```

- Onde <nome_tabela> é a tabela que se deseja excluir do banco de dados.

CUIDADO!!!


VIDADEPROGRAMADOR
.COM.BR
/* HISTÓRIA REAL
ENVIADA POR
DIEGO ANTONIO SALES */
#534
CARA, O QUE É "DROP"?
SIGNIFICA APAGAR/EXCLUIR/ DELETAR...
NÃO SABE NÃO USE
AHHHH, DAÍ JOGA NUMA LIXEIRA?
NÃO... ELE APAGA DIRETO E JÁ ERA... DEPOIS SÓ COM BACKUP..
HMMM... VOCÊ TEM AÍ UM BACKUP DA TABELA DE PRODUTOS?
PLOFT!

Manipulando os dados de um banco de dados

Remoção

- Um pedido para remoção de dados é expresso muitas vezes do mesmo modo que uma consulta;
- Pode-se remover somente tuplas inteiras;
- Em SQL a remoção é expressa por:

DELETE FROM tabela WHERE predicado;

Remoção

- Exemplo:
  - Remover todos os empréstimos com total entre 1300 e 1500 reais:

DELETE FROM emprestimo WHERE total BETWEEN 1300 AND 1500;

Inserção

- Para inserir dados em uma tabela podemos especificar uma tupla a ser inserida ou escrever uma consulta cujo o resultado é a um conjunto de tuplas a serem inseridas;
- A inserção é expressa por:

INSERT INTO tabela(A1, A2) VALUES (V1, V2);

- Onde Ai representa os atributos a serem inseridos e Vi os valores.

Inserção

- Exemplo:
  - Deve-se inserir a informação que existe um conta numero 25 na agencia “Agencia13” e que ela tem um saldo de 199 reais:

INSERT INTO conta (numero_conta, nome_agencia, saldo) VALUES (25, ‘Agencia13’, 199);

Inserção

- A instrução ***insert*** pode conter subconsultas que definem os dados a serem inseridos:

INSERT INTO tabela1 (A1, A2) SELECT (A1, A2) FROM tabela2;

- Com isto a instrução insert insere mais de uma tupla.

Atualizações

- Em determinadas situações, pode-se querer modificar valores das tuplas.
- Para alterações utiliza-se o comando ***UPDATE***.

UPDATE tabela SET A1 = V1, A2 = V2 WHERE predicado;

- Onde ***Ai*** representa o atributo em questão e ***Vi*** o valor alterado.

Atualizações

- Exemplo:
  - Suponha que o pagamento da taxa de juros anual esteja sendo efetuado e todos os saldos deverão ser atualizados de em 5%:

UPDATE conta SET saldo = saldo *1,05;

Atualizações

- Exemplo:
  - Suponha que agora contas com saldo superior a 10000 reias, recebam 6 por cento de juros:

UPDATE conta SET saldo = saldo * 1,06 WHERE saldo > 10000;

Campo serial

- O campo serial é uma alternativa para a criação de campos auto-incrementáveis:

CREATE TABLE users (id SERIAL PRIMARY KEY, name TEXT, age INT4);

Campo serial

- Para inserir valores na tabela criada no slide anterior:

INSERT INTO users (name, age) VALUES ('Mozart', 20);

OU

INSERT INTO users (name, age, id) VALUES ('Mozart', 20, DEFAULT);

Recuperando dados

Estrutura Básica

- Uma consulta típica em SQL tem a seguinte forma:

SELECT a1, a2, a3, ..., an FROM r1, r2, ..., rn WHERE P;

  - Cada ***ai*** representa um atributo de cada ***ri*** (relação), P é o predicado (condição).
  - A consulta é equivalente a seguinte expressão em álgebra relacional:

$$\pi_{A1, A2, .., An} (\sigma_{p} (R1 \times R2 \times \ldots Rn))$$

Cláusula SELECT

Cláusula SELECT

- Exemplo: Encontre os nomes de todas as agencias da relação empréstimo:

  SELECT nome_agencia FROM emprestimo;

- No caso em que desejamos a forçar a eliminação, podemos inserir a palavra chave ***distinct*** depois do select.

SELECT DISTINCT nome_agencia FROM emprestimo;

Cláusula SELECT

- O asterisco '*' pode ser usado para denotar "todos os atributos", assim para exibir todos os atributos de empréstimo:

SELECT * FROM emprestimo;

- Select pode conter expressões aritméticas envolvendo operadores: +,-,/ e *:

SELECT nome_agencia, total * 100 FROM emprestimo;

Cláusula WHERE

Cláusula WHERE

- Considere a consulta:
  - "encontre todos os números de empréstimos feitos na agência "Central" com totais de empréstimos acima de 1200 dólares":
  - Esta consulta pode ser escrita em SQL como:

SELECT numero_emprestimo FROM emprestimo
WHERE nome_agencia = 'Central'
AND total>1200;

Cláusula WHERE

- SQL usa conectores lógicos **and**, **or** e **not** na cláusula **where**;

- Os operadores dos conectivos lógicos podem ser expressões envolvendo operadores de comparação: <, <=, >, >= e <>

Cláusula WHERE

- SQL possui operador de comparação **between** para simplificar a cláusula **where** que especifica valores dentro de um intervalo.
  - Se desejarmos saber os números de empréstimos cujo o montante esteja entre 90000 e 100000, pode-se fazer:

SELECT numero_emprestimo FROM emprestimo WHERE total BETWEEN 90000 AND 100000;

Cláusula FROM

Cláusula FROM

- A cláusula **from** por sí só define um **produto cartesiano** das relações da cláusula.

Cláusula FROM

Expressão sem produto cartesiano:

$$(\pi_{\text{nome\_cliente, numero\_emprestimo}}(\sigma_{\text{devedor.numero\_emprestimo=emprestimo.numero\_emprestimo}}(\text{devedor} \times \text{emprestimo})))$$

Para consultar todos os clientes que tenham um empréstimo em um banco, encontre seus nomes é números de empréstimo, em SQL pode ser escrito:

SELECT DISTINCT nome_cliente, devedor.numero_emprestimo FROM devedor, emprestimo WHERE devedor.numero_emprestimo = emprestimo.numero_emprestimo;

Cláusula FROM

- ***Importante***: na consulta anterior usa-se a notação ***nome_relação.nome_atributo***, como na álgebra relaciona para evitar ambiguidade nos casos em que um atributo apareça em mais de uma relação.

Cláusula FROM

- Para impedir a formação de um produto cartesiano, sempre inclua uma condição de restrição válida na cláusula ***WHERE***.
  - ***Relembrando produto cartesiano:***

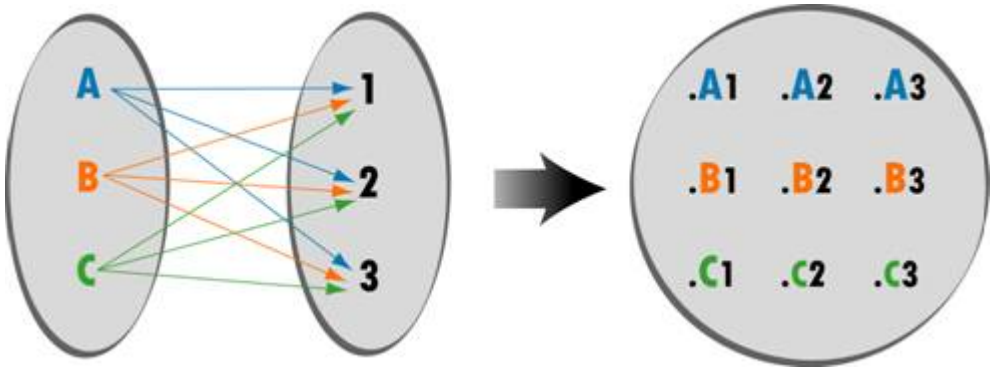
Clark	10
Miller	10
Smith	20
Turner	30

PRODUTO

10	Accounting
20	Research
30	Sales

=

Clark	10	10	Accounting
Clark	10	20	Research
Clark	10	30	Sales
Miller	10	10	Accounting
Miller	10	20	Research
Miller	10	30	Sales
Smith	20	10	Accounting
Smith	20	20	Research
Smith	20	30	Sales
Turner	30	10	Accounting
Turner	30	20	Research
Turner	30	30	Sales



Cláusula FROM

- Comparação
  - Com produto cartesiano:

Clark	10
Miller	10
Smith	20
Turner	30

PRODUTO

10	Accounting
20	Research
30	Sales

=

Clark	10	10	Accounting
Clark	10	20	Research
Clark	10	30	Sales
Miller	10	10	Accounting
Miller	10	20	Research
Miller	10	30	Sales
Smith	20	10	Accounting
Smith	20	20	Research
Smith	20	30	Sales
Turner	30	10	Accounting
Turner	30	20	Research
Turner	30	30	Sales

  - Com Junção:

Clark	10
Miller	10
Smith	20
Turner	30

JOIN

10	Accounting
20	Research
30	Sales

=

Clark	10	10	Accounting
Miller	10	10	Accounting
Smith	20	20	Research
Turner	30	30	Sales

Variáveis Tuplas

- Variáveis tuplas são definidas na cláusula **FROM** por meio do uso da cláusula **AS**.
- A seguir é apresentada a consulta "para todos os funcionários encontre o nome do seu supervisor":

SELECT DISTINCT f.nome_funcionario, s.nome_funcionario FROM funcionario AS f, funcionario AS s WHERE f.rg_funcionario = s.rg_funcionario;

Operações em String

- As operações em Strings mais usadas são as checagens para verificação de coincidências de pares, usando o operador **like**.

- Indicaremos esses pares por meio do uso de caracteres especiais:
  - Porcentagem (**%**): compara qualquer substring;
  - Sublinhado (**_**): compara qualquer caracter.

Operações em String

- Comparações desse tipo são sensíveis ao tamanho das letras; isto é, minúsculas não são iguais a maiúsculas e vice-versa;
- Para ilustrar considere os seguintes exemplos:
  - ‘Pedro%’ corresponde a qualquer String que comece com Pedro.
  - ‘%inh%’ corresponde a qualquer String que possua uma substring ‘inh’, por exemplo: ‘huguinho’, ‘zezinho’ e ‘luizinho’.

Operações em String

- Pares são expressões em SQL usando o operador de comparação ***LIKE***. Considere a consulta: “encontre os nomes de todos os clientes que possuam ‘Silva’ ”:

  SELECT nome_cliente FROM cliente WHERE nome_cliente LIKE ‘%Silva%’;

SQL permite pesquisar diferenças em vez de coincidências usando ***NOT LIKE***.

Ordenação e Apresentação de Tuplas

- SQL oferece ao usuário algum controle sobre a ordenação por meio da qual as tuplas de uma relação serão apresentadas.

- A cláusula ***ORDER BY*** faz com que as tuplas do resultado de uma consulta apareçam em uma determinada ordem.

Ordenação e Apresentação de Tuplas

- Para listar em ordem alfabética todos os clientes que tenham um empréstimo na agência Centro:

SELECT DISTINCT nome_cliente FROM devedor AS d, emprestimo AS e WHERE d.numero_emprestimo = e.numero_emprestimo AND nome_agencia = 'Centro' ORDER BY nome_cliente;

Ordenação e Apresentação de Tuplas

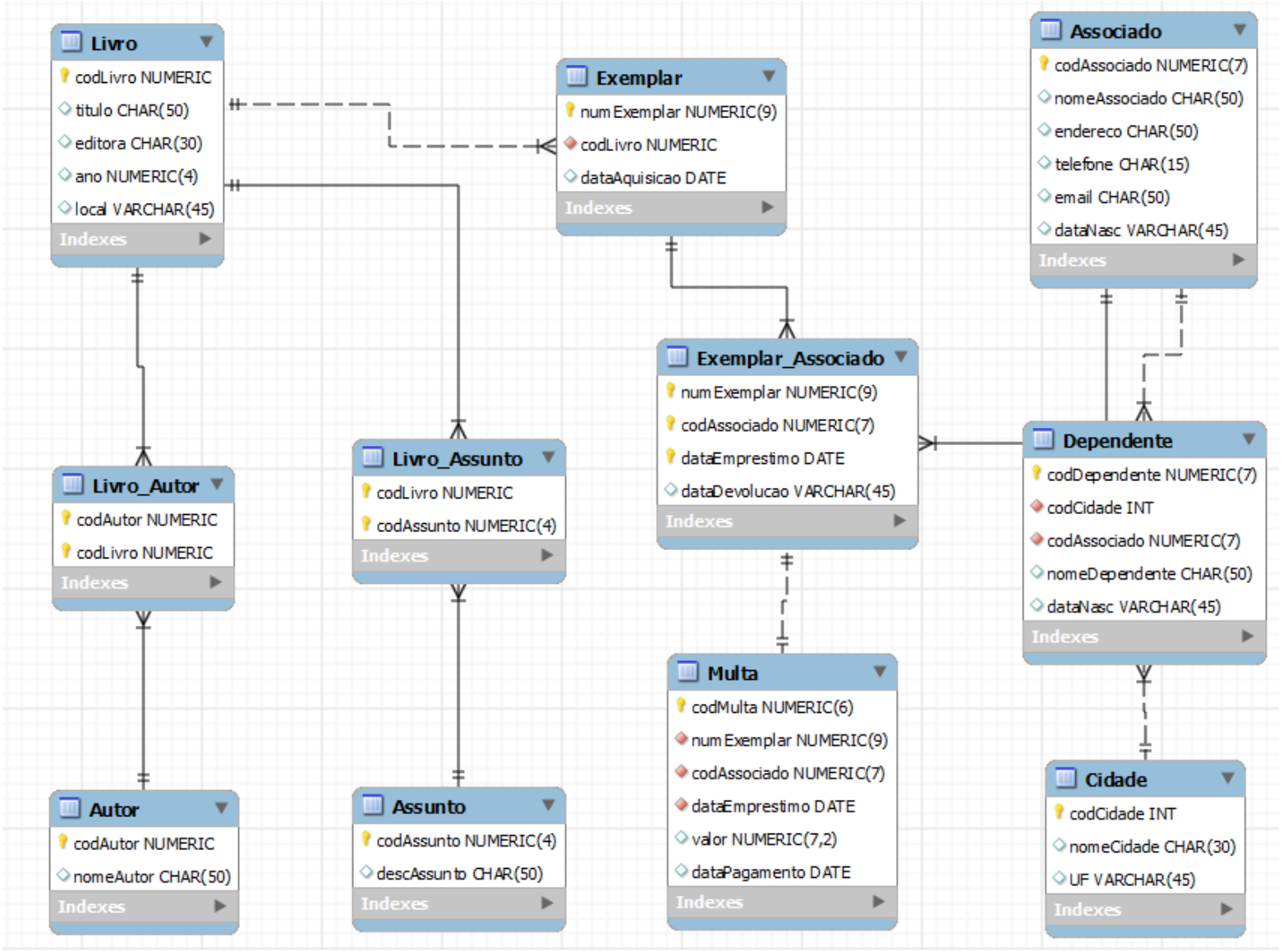
- Por default a cláusula ***ORDER BY***, relaciona os itens em ordem acedente (***ASC***), para inverter a ordem pode-se utilizar ***DESC***.

SELECT cidade_cliente, nome_cliente FROM cliente ORDER BY cidade_cliente ASC, nome_cliente DESC;

Exercícios


VIDADEPROGRAMADOR
.COM.BR
/* HISTÓRIA REAL ENVIADA POR FERNANDO */
BY NC SA
#568
AQUI É DO SUPORTE... EU ARRUMEI UM ERRO NA GERAÇÃO DO ARQUIVO, SÓ QUE AGORA TEM UM ERRO NOVO... ESTOU ENCAMINHANDO PRA VOCÊ VERIFICAR.
NÃO TOQUE NO MEU CÓDIGO
CERTO... SÓ ME DIGA O QUE VOCÊ FEZ E QUAL O ERRO NOVO...
NÃO TOQUE NO MEU CÓDIGO
EU COMENTEI UM SELECT E O ERRO NÃO ACONTECEU MAIS, SÓ QUE AGORA O ARQUIVO GERA EM BRANCO...
PLOFT!

Modelo base do Exercício



Exercício 1

- Deve-se montar o SCRIPT de criação das tabelas apresentadas no modelo anterior.

- As chaves amarelas significam que o campo é chave primária.
- Quando o losango é vermelho, quer dizer que é uma chave estrangeira.

Exercício 2

- Levando em consideração o modelo anterior, desenvolva uma consulta para resolver a seguinte questão:
  - Quais livros foram publicados a partir de 2008?
  - Apresente como resultado os seguintes atributos:
    - Nome do livro;
    - Nome do autor;
    - Ano de publicação.

Exercício 3

- A partir da consulta gerada no Exercício 1, também é desejado receber como resultado da consulta o seguinte atributo:

  – Assunto do livro.

Exercício 4

- Apresentar todos os livros emprestados para o associado "José da Silva“.
- A consulta deve apresentar os seguintes valores:
  - Nome do associado;
  - Título do livro;
  - Data do empréstimo;
  - Data da devolução;

Exercício 5

- A biblioteca gostaria de saber quais assuntos o associado "José da Silva" tem interesse.
- A consulta deve retornar os seguintes valores:
  - Assunto (sem repetição);

Exercício 6

- Deve-se apresentar todas as multas por ordem crescente de valor do associado "José da Silva”
- A consulta deve retornar os seguintes valores:
  - Título do livro relacionado com a multa;
  - Data do empréstimo;
  - Valor da multa;

Exercício 7

- Apresentar todos os associados que pagaram mais de R$5,00 de multa.
- A consulta deve retornar os seguintes valores:
  - Nome do associado;
  - Assunto do livro;
  - Valor da multa;
  - Data de pagamento.

Exercício 8

- Apresentar o nome de todos os associados que possuem multas em aberto.
- A consulta deve retornar os seguintes valores:
  - Nome do associado;
  - Telefone
  - Título do livro
  - Data empréstimo
  - Valor da multa.