



# Banco de Dados

# Objetos Persistentes

- **Objetos Persistentes:** são objetos que requerem armazenamento persistente.

Exemplo: Instâncias da classe Descrição Produto devem ser armazenadas em uma base de dados.

# Mecanismos de Armazenamento de Objetos

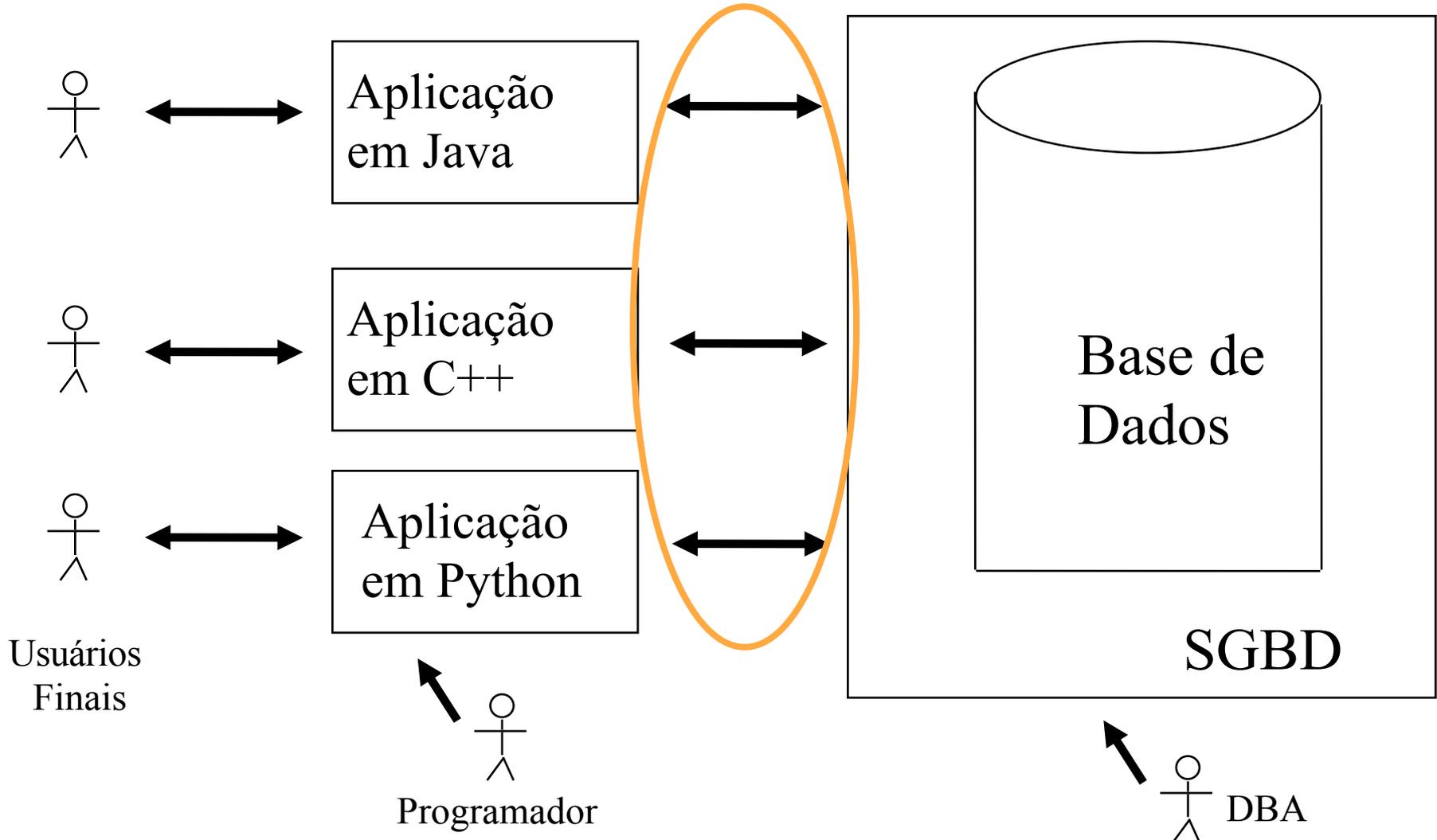
- Banco de Dados Orientado a Objetos

↳ não requer mapeamento para armazenar e recuperar objetos

- ➡ • Banco de Dados Relacional

- Outros: Banco de Dados Hierárquico, Arquivo, XML, etc.

# Armazenamento em um BD Relacional



# Modelo Relacional

No modelo relacional, a base de dados é descrita através de relações.

Informalmente, uma relação pode ser vista como uma tabela.

**Exemplo:** Empregados

Cód.	Nome	Salário
110	Benedito Gomes	567,43
200	Alberto Silva	1.345,00
835	João Santos	880,00
902	Cláudia Vieira	600,00

Cada tupla (linha) representa uma coleção de valores de dados relacionados.

O nome da relação (tabela) e os nomes dos atributos (colunas) são usados para ajudar na interpretação do significado dos valores em cada tupla (linha).

# Modelo Relacional

## RELAÇÃO

*versus*

## TABELA

Atributos

Colunas

Tuplas

Linhas

Deve ter chave primária

Não precisa de chave primária

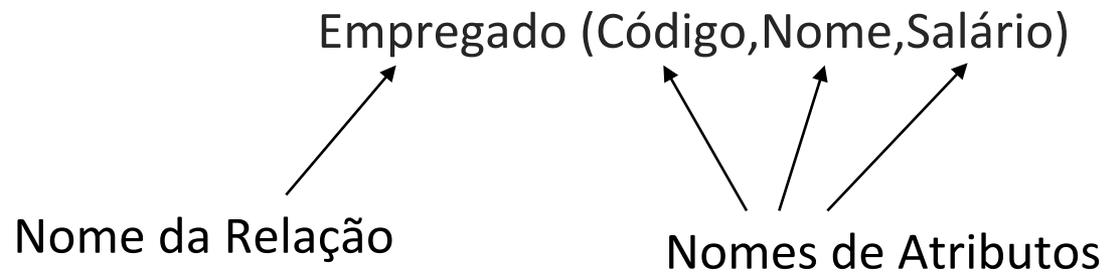
É um conjunto, i.e., não pode conter elementos repetidos

Pode conter elementos repetidos

# Conceitos do Modelo Relacional

A estrutura de uma relação é definida por um conjunto de atributos.

O nome de uma relação e a lista de seus atributos definem o **esquema da relação** (intensão).

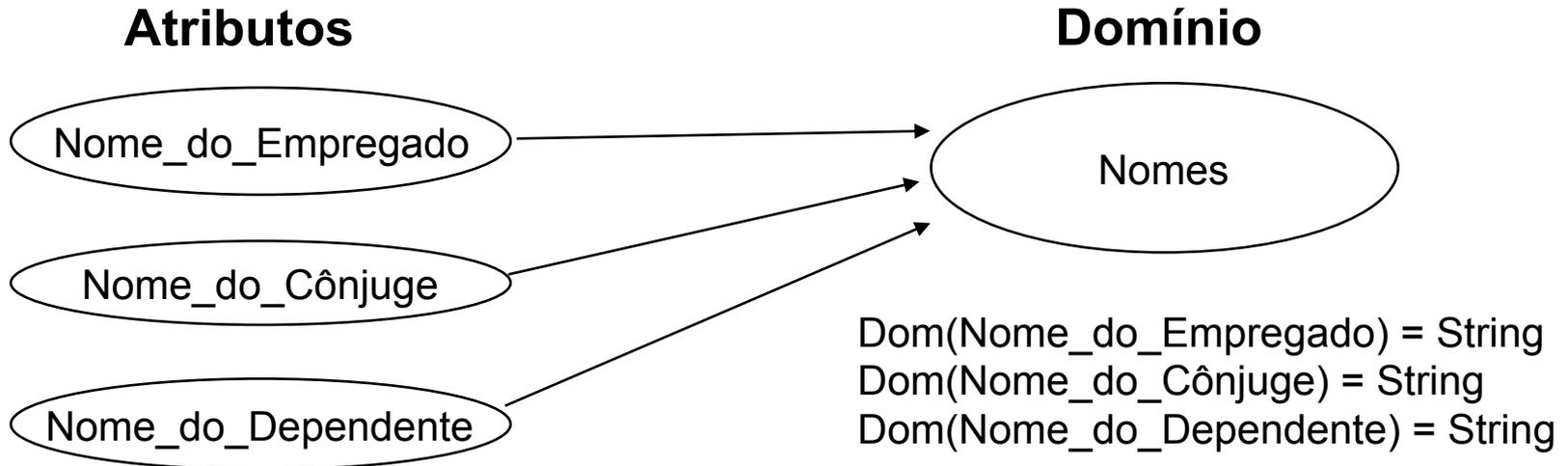


Exemplos de Esquemas de Relação:

- ALUNO (Nome, Matricula, Fone, Endereco, Idade)
- ALUNO (Nome: string, Matricula: string, Fone: string, Endereco: string, Idade: string)

# Conceitos do Modelo Relacional

- **Atributo:** É o nome do papel assumido por um domínio no esquema de uma relação



- **Domínio:** É um conjunto de valores atômicos (indivisíveis) permitidos para um dado.

Exemplos:

**String** - o conjunto de todas as cadeias de caracteres

**Inteiro** - o conjunto de todos os números inteiros

# Conceitos do Modelo Relacional

➤ Formalmente, uma relação pode ser definida como:

Dado um esquema  $R(A_1, A_2, \dots, A_n)$ , uma **relação**  $r(R)$  é um conjunto de tuplas  $r = \{t_1, t_2, \dots, t_m\}$ , onde cada tupla é uma lista de  $n$  valores  $t = \langle v_1, v_2, \dots, v_n \rangle$ , sendo que cada valor  $v_i$  é um elemento de  $\text{Dom}(A_i)$ , ou é igual a **null** (constante especial que indica a ausência de valor).

Obs: A ordem das tuplas não é relevante.

# Conceitos do Modelo Relacional

- **Chave:** é o conjunto **mínimo** de atributos cujos valores identificam uma tupla na relação.

Exemplos:

- alunos: **matrícula** (chave 1), **cpf**(chave 2)
- cidades: (**nome, estado**)
- Uma relação pode ter mais de uma chave. Cada chave é denominada **chave candidata**. Uma das chaves candidatas é escolhida para ser a **chave primária (PK)** (aparece sublinhada no esquema da relação)

Empregado ( Código, Nome, Salário )

# Conceitos do Modelo Relacional

- **Chave Estrangeira (FK):** é um conjunto de atributos (e respectivos domínios) cujos valores devem se referir à chave primária de uma relação.

Exemplos:

- alunos: **curso** (referência a um código de curso)

- cursos: **código**

- Se  $fk$  é uma chave estrangeira em  $R_1$  que faz referência à chave primária  $pk$  de  $R_2$  então:

$$\text{domínio}(fk) = \text{domínio}(pk)$$

$R_1$  e  $R_2$  podem ser a mesma relação.

# Restrições no Modelo Relacional

- **1 - Restrições de Domínio:** cada valor de um atributo A deve ser um valor atômico pertencente a  $Dom(A)$ .

$Dom(\text{Salário}) = \text{Reais}$

<u>Código</u>	Nome	Salário
110	Benedito Gomes	567,43
200	Alberto Silva	345,00
835	João Santos	280,00
902	Cláudia Vieira	200,00
005	Walter Souza	1456,86
711	Flávia Costa	450,00
521	Maria da Silva	87,32

# Restrições no Modelo Relacional

- **2 - Restrições de Chave:** Tuplas pertencentes a uma mesma relação não podem possuir o mesmo valor para uma mesma **chave**.

<u>Código</u>	Nome	Salário
110	Benedito Gomes	567,43
200	Alberto Silva	345,00
835	João Santos	280,00
902	Cláudia Vieira	200,00
005	Walter Souza	1456,86
711	Flávia Costa	450,00
521	Maria da Silva	87,32

# Restrições no Modelo Relacional

- **3 - Restrição de Integridade de Entidade:** O valor dos atributos que compõem a chave primária **não pode ser null** para nenhuma tupla da relação.



<u>Código</u>	Nome	Salário
110	Benedito Gomes	567,43
200	Alberto Silva	345,00
835	João Santos	280,00
902	Cláudia Vieira	200,00
005	Walter Souza	1456,86
711	Flávia Costa	450,00
521	Maria da Silva	87,32

# Restrições no Modelo Relacional

- **4 - Restrição de Integridade Referencial:** Todo valor de chave primária referenciado por uma **chave estrangeira** (FK) deve obrigatoriamente existir na relação associada (exceto quando este valor é **null**)

## Empregado

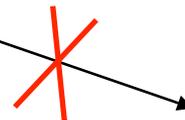
<u>Código</u>	Nome	Depto(FK)
110	Benedito Gomes	12
200	Alberto Silva	19
835	João Santos	<null>
902	Cláudia Vieira	45

## Departamento

<u>Código</u>	Nome
12	Vendas
15	Marketing
08	Finanças
19	Produção

OK!

OK!



- Garante que todos os relacionamentos sejam válidos.

# SQL (Structured Query Language)

- A estrutura de dados manipulada pela SQL é a TABELA
- É utilizada em SGBDs relacionais como
  - DDL (Data Definition Language) - definição de tabelas
  - DML (Data Manipulation Language) - consulta e atualização

# SQL como DDL

➤ Comandos para definição de esquemas:

➔ **create table**

define a estrutura da tabela, suas restrições de integridade e cria uma tabela vazia

➔ **alter table**

modifica a definição de uma tabela

➔ **drop table**

remove uma tabela com todas as suas linhas

# SQL como DDL - CREATE TABLE

- A criação de tabelas é feita através do comando CREATE TABLE

```
CREATE TABLE <nome_da_tabela>
( <nome_da_col1>    <tipo_da_col1>    [ NOT NULL ] [ UNIQUE ],
  <nome_da_col2>    <tipo_da_col2>    [ NOT NULL ] [ UNIQUE ],
  ...
  [ PRIMARY KEY (<nome_col1>, <nome_col2>, ...) ]
  [ FOREIGN KEY (<nome_col1>, <nome_col2>, ...)
    REFERENCES
    <nome_tab_ref>(<nom_col1_ref>, <nom_col2_ref>, ...) ]
)
```

Nota: [ ] - cláusula opcional

# Exemplo - CREATE TABLE

- Exemplo de criação de tabelas

```
CREATE TABLE FORNECEDOR
( F          INTEGER          NOT NULL,
  FNAME     VARCHAR(30)      NOT NULL,
  STATUS    INTEGER,
  CIDADE    VARCHAR(20)
)
```

```
CREATE TABLE FORNECEDOR
( F          INTEGER,
  FNAME     VARCHAR(30)      NOT NULL,
  STATUS    INTEGER,
  CIDADE    VARCHAR(20) ,
  PRIMARY KEY (F)
)
```

# Exemplo - CREATE TABLE

- Exemplo de criação de tabelas com chave primária composta e chave estrangeira

```
CREATE TABLE EMPREGADO
( NOME          VARCHAR(40),
  CPF           VARCHAR(15),
  SALARIO      DECIMAL(7,2),
  COD_DEPT     INTEGER    NOT NULL,
  PRIMARY KEY  (NOME, CPF),
  FOREIGN KEY  (COD_DEPT) REFERENCES DEPARTAMENTO(COD)
)
```

# SQL como DDL - ALTER TABLE

- A alteração de tabelas é feita através do comando ALTER TABLE

```
ALTER TABLE <nome_da_tabela> <alteração>;
```

A alteração pode ser:

1. **ADD** [COLUMN] <nome\_da\_col1> <tipo\_da\_col1> [{RIs}]  
[ {, <nome\_da\_colN> <tipo\_da\_colN> [{RIs}]} ]
2. **MODIFY** [COLUMN] <nome\_da\_col1> <tipo\_da\_col1> [{RIs}]  
[ {, <nome\_da\_colN> <tipo\_da\_colN> [{RIs}]} ]
3. **DROP COLUMN** <nome\_da\_col1>  
[ {, <nome\_da\_col1> } ]
4. **ADD CONSTRAINT** nome\_RI\_1 def\_RI\_1  
[ {, nome\_RI\_n def\_RI\_n } ]
5. **DROP CONSTRAINT** nome\_RI\_1  
[ {, nome\_RI\_n } ]
6. [ **ADD** | **DROP** ] [ **PRIMARY KEY** | **FOREIGN KEY** nome\_RI\_1 ]

# SQL como DDL - ALTER TABLE

```
ALTER TABLE FORNECEDOR ADD ENDERECO VARCHAR(30)
```

```
ALTER TABLE FORNECEDOR DROP PRIMARY KEY (MySQL)
```

```
ALTER TABLE FORNECEDOR DROP CONSTRAINT fornecedor_pkey (PostgreSQL)
```

```
ALTER TABLE FORNECEDOR DROP COLUMN ENDERECO, DROP COLUMN CIDADE
```

```
ALTER TABLE PECA ADD FOREIGN KEY(FORNEC) REFERENCES FORNECEDOR(F)
```

```
ALTER TABLE PECA ADD constraint fk_fornecedor FOREIGN KEY(FORNEC)  
REFERENCES FORNECEDOR(F)
```

```
ALTER TABLE PECA DROP FOREIGN KEY PECA_ibfk_1 (MySQL)
```

```
ALTER TABLE PECA DROP CONSTRAINT peca_fornecedor_fkey (PostgreSQL)
```

# SQL - DROP TABLE

- A destruição de tabelas é feita através do comando DROP TABLE

```
DROP TABLE <nome_da_tabela>
```

- Exemplo

```
DROP TABLE PECA
```

# Exemplo - Esquema Relacional

## Ambulatorio

<u>Num_a</u>	Andar	Capacidade
--------------	-------	------------

## Medico

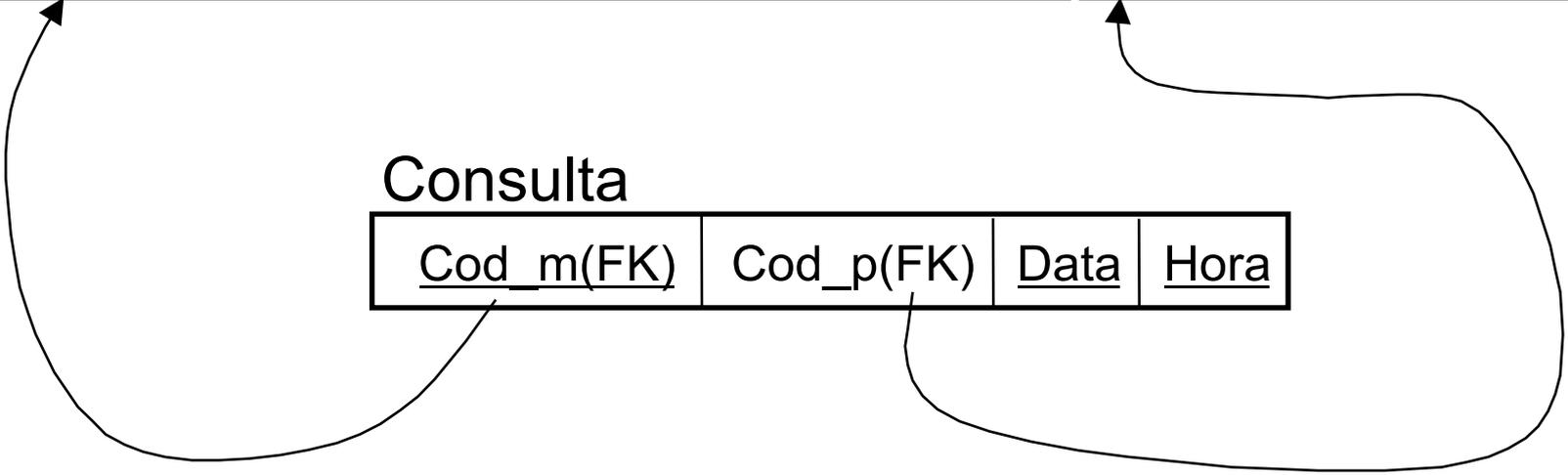
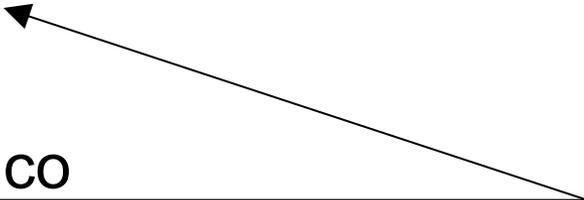
<u>Cod_p</u>	Nome	Idade	Espec	Cidade	Num_a (FK)
--------------	------	-------	-------	--------	------------

## Paciente

<u>Cod_p</u>	Nome	Idade	Cidade
--------------	------	-------	--------

## Consulta

<u>Cod_m(FK)</u>	Cod_p(FK)	<u>Data</u>	<u>Hora</u>
------------------	-----------	-------------	-------------



# SQL - Exercício

1) Através do comando CREATE TABLE, crie as tabelas correspondentes às relações abaixo:

AMBULATORIO (num\_a : int, andar : numeric(3), capacidade : smallint)

MEDICO (cod\_m : int, nome : varchar(40), idade : smallint,  
especialidade : char(20), cidade : varchar(30), num\_a (FK): int)

PACIENTE (cod\_p : int, nome : varchar(40), idade : smallint,  
cidade : varchar(30))

CONSULTA (cod\_m (FK) : int, cod\_p (FK) : int, data : date, hora : time)

(Copiar parte dos comandos do arquivo “Inicializacao-Tabelas” disponível no endereço [www.inf.ufsc.br/~vilain/aula-es-bd](http://www.inf.ufsc.br/~vilain/aula-es-bd))

# SQL como DML

Define operações de manipulação de dados

- INSERT
- UPDATE
- DELETE
- SELECT

# SQL - INSERT

- A inserção de dados nas tabelas é feita através do comando INSERT

```
INSERT INTO <nome_da_tabela>  
[(<nome_da_col1>, <nome_da_col2>,...)]  
VALUES (valor_da_col1, valor_da_col2,...)
```

# Exemplo - INSERT

- Exemplos de inserção de dados

```
INSERT INTO FORNECEDOR  
VALUES (1, 'Smith', 20, 'Londres')
```

que equivale a:

```
INSERT INTO FORNECEDOR (F, FNOME, STATUS, CIDADE)  
VALUES (1, 'Smith', 20, 'Londres')
```

Também é possível especificar somente algumas colunas:

```
INSERT INTO FORNECEDOR (FNOME, F)  
VALUES ('Smith', 1)
```

que equivale a:

```
INSERT INTO FORNECEDOR (F, FNOME, STATUS, CIDADE)  
VALUES (1, 'Smith', null, null)
```

# Exercício - INSERT

- Insira os seguintes dados na tabela **Ambulatorio**

<u>Num_a</u>	Andar	Capacidade
1	1	30
2	1	50
3	2	40
4	2	25
5	2	55
6	2	10
7	2	10

(Copiar parte dos comandos do arquivo “Inicializacao-Tabelas” disponível no endereço [www.inf.ufsc.br/~vilain/aula-es-bd](http://www.inf.ufsc.br/~vilain/aula-es-bd))

# Exercício - INSERT

- Insira os seguintes dados na tabela **Medico**

<u>Cod_m</u>	Nome	Idade	Especialidade	Cidade	Num_a
1	'Joao'	40	'ortopedista'	'Florianopolis'	1
2	'Maria'	42	'oftalmologista'	'Blumenau'	2
3	'Pedro'	51	'pediatra'	'Sao Jose'	2
4	'Carlos'	28	'ortopedista'	'Florianopolis'	4
5	'Marcia'	33	'neurologista'	'Florianopolis'	3
6	'Pedrinho'	38	'infectologista'	'Blumenau'	1
7	'Mariana'	39	'infectologista'	'Florianopolis'	NULL
8	'Roberta'	50	'neurologista'	'Joinville'	5
9	'Vanusa'	39	'aa'	'Curitiba'	NULL
10	'Valdo'	50	'aa'	'Curitiba'	NULL

# Exercício - INSERT

- Insira os seguintes dados na tabela **Paciente**

<u>Cod_p</u>	Nome	Idade	Cidade
1	'Clevi'	60	'Florianopolis'
2	'Cleusa'	25	'Palhoca'
3	'Alberto'	45	'Palhoca'
4	'Roberta'	44	'Sao Jose'
5	'Fernanda'	3	'Sao Jose'
6	'Luanda'	2	'Florianopolis'
7	'Manuela'	69	'Florianopolis'
8	'Caio'	45	'Florianopolis'
9	'Andre'	83	'Florianopolis'
10	'Andre'	21	'Florianopolis'

# Exercício - INSERT

- Insira os seguintes dados na tabela **Consulta**

<u>Cod_m</u>	<u>Cod_p</u>	<u>Data</u>	<u>Hora</u>
1	3	'2000/06/12'	'14:00'
4	3	'2000/06/13'	'9:00'
2	9	'2000/06/14'	'14:00'
7	5	'2000/06/12'	'10:00'
8	6	'2000/06/19'	'13:00'
5	1	'2000/06/20'	'13:00'
6	8	'2000/06/20'	'20:30'
6	2	'2000/06/15'	'11:00'
6	4	'2000/06/15'	'14:00'
7	2	'2000/06/10'	'19:30'

# SQL - UPDATE

- A modificação de dados nas tabelas é feita através do comando UPDATE.

```
UPDATE <nome_tabela>  
SET <lista_de_atribuição>  
WHERE <condição>
```

- Exemplo:

```
UPDATE PROJETO  
SET JNOME = 'Deck',  
    CIDADE = 'Brasilia'  
WHERE J = 7
```

# Exercícios

- Modifique a cidade dos médicos de Blumenau para Biguaçu.
- Modifique a cidade dos médicos de Biguaçu para Blumenau .
- Modifique a capacidade do ambulatório 1 para 35.

# SQL - DELETE

- A remoção de dados nas tabelas é feita através do comando DELETE.

```
DELETE FROM <nome_tabela>  
[ WHERE <condição> ]
```

- Exemplos:

```
DELETE FROM PROJETO
```

```
DELETE FROM PECA  
WHERE P = 2
```

# Exercício

- Remova todos os médicos de Curitiba.
- Remova todos os ambulatórios com capacidade 10.

# SQL - SELECT

- A consulta nas tabelas é feita através do comando SELECT

```
SELECT <lista_de_colunas>  
FROM <lista_de_tabelas>  
WHERE <condição>
```

onde:

<lista\_de\_colunas> - nomes das colunas cujos valores serão retornados pela consulta

<lista\_de\_tabelas> - nomes das tabelas utilizadas na consulta

<condição> - expressão lógica que determina as linhas que serão retornadas pela consulta

# Exemplo - SELECT

- Obter o conteúdo da tabela de peças

```
SELECT P, PNOOME, COR, PESO, CIDADE  
FROM PECA
```

P	PNOOME	COR	PESO	CIDADE
1	Parafuso	Vermelho	12	Sao Paulo
2	Porca	Verde	17	Porto Alegre
3	Martelo	Azul	17	Rio
4	Martelo	Vermelho	14	Sao Paulo
5	Prego	Azul	12	Rio
6	Broca	Vermelho	19	Sao Paulo

# Exemplo - SELECT

- Obter o conteúdo da tabela de peças

```
SELECT * FROM PECA
```

Equivalente a  
"todos os atributos"



P	PNOME	COR	PESO	CIDADE
1	Parafuso	Vermelho	12	Sao Paulo
2	Porca	Verde	17	Porto Alegre
3	Martelo	Azul	17	Rio
4	Martelo	Vermelho	14	Sao Paulo
5	Prego	Azul	12	Rio
6	Broca	Vermelho	19	Sao Paulo

# SQL - SELECT

- O que é uma **álgebra**? Uma álgebra é definida por uma estrutura de dados e por uma coleção de operações aplicáveis a esta estrutura.

A **álgebra relacional** é caracterizada por operações aplicáveis a relações.

- Algumas operações da álgebra relacional utilizadas no SELECT:
  - Seleção  $\sigma$
  - Projeção  $\pi$
  - Produto Cartesiano  $\times$
  - Junção  $\otimes$

# SQL - SELECT

- Algumas operações da álgebra relacional utilizadas no SELECT:

- Seleção  $\sigma$
- Projeção  $\pi$
- Produto Cartesiano  $\times$
- Junção  $\otimes$

Notação da Seleção:  $\sigma_{\langle \text{predicado} \rangle} (R)$

Operadores de comparação:  $=, <, <=, >, >=, \neq$

Operadores lógicos:  $\wedge$  (and)  $\vee$  (or)  $\neg$  (not)

# Exemplo - SELECT

- Obter o conteúdo de todas as peças denominadas 'Martelo'

```
SELECT * FROM PECA
WHERE PNOME = 'Martelo'
```

 **SELEÇÃO** da  
álgebra relacional

P	PNOME	COR	PESO	CIDADE
3	Martelo	Azul	17	Rio
4	Martelo	Vermelho	14	Sao Paulo

# SQL - SELECT

- Algumas operações da álgebra relacional utilizadas no SELECT:
  - Seleção  $\sigma$
  - **Projeção  $\pi$**
  - Produto Cartesiano  $\times$
  - Junção  $\bowtie$

Notação da Projeção:  $\pi_{\langle \text{lista de atributos} \rangle} (R)$

# Exemplo - SELECT

- Obter o nome e a cor de cada peça

```
SELECT PNOOME, COR
FROM PECA
```

**PROJEÇÃO** da  
álgebra relacional



PNOOME	COR
Parafuso	Vermelho
Porca	Verde
Martelo	Azul
Martelo	Vermelho
Prego	Azul
Broca	Vermelho

# Exemplo - SELECT

- Obter as cidades dos fornecedores

```
SELECT CIDADE
FROM FORNECEDOR
```

```
CIDADE
```

```
=====
```

```
Rio
```

```
Sao Paulo
```

```
Porto Alegre
```

```
Rio
```

```
Curitiba
```

```
Curitiba
```

Pergunta:

Qual o problema no resultado desta consulta?

# Exemplo – Cláusula DISTINCT

- Obter as cidades dos fornecedores

```
SELECT DISTINCT CIDADE  
FROM FORNECEDOR
```

CIDADE

=====

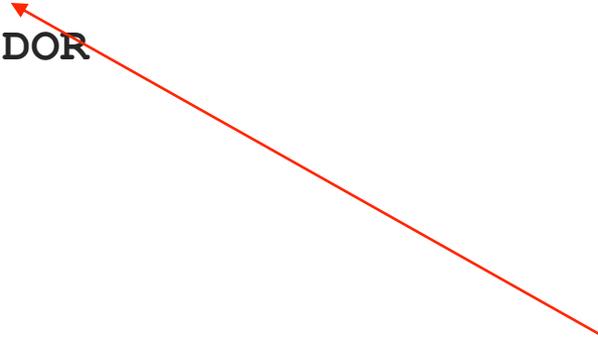
Rio

Sao Paulo

Porto Alegre

Curitiba

A cláusula DISTINCT serve para eliminar as repetições



# Exercícios

- Obter os médicos de Florianópolis.
- Obter o código do médico com nome Marcia.
- Obter as especialidades dos médicos de Florianópolis, eliminando as repetições.

# SQL - SELECT

- Algumas operações da álgebra relacional utilizadas no SELECT:
  - Seleção  $\sigma$
  - Projecção  $\pi$
  - Produto Cartesiano  $\times$
  - Junção  $\bowtie$

Notação do Produto Cartesiano: relação1  $\times$  relação2

Notação Junção: relação1  $\bowtie$  condição de junção relação2

# Exemplo - Cláusula FROM

- Todas as tabelas que aparecem na cláusula FROM são combinadas
- Se não houver cláusula WHERE, o resultado é o produto cartesiano das tabelas

```
SELECT * FROM FORNECEDOR, FORNECIMENTO
```

# Exemplo - Cláusula FROM

F	FNOME	STATUS	CIDADE	F	P	QUANTIDADE
1	Antonia	20	Rio	1	1	200
2	Mariana	10	Sao Paulo	1	1	200
3	Juliana	30	Porto Alegre	1	1	200
4	Maria	20	Rio	1	1	200
5	Josiane	30	Curitiba	1	1	200
6	Maria	30	Curitiba	1	1	200
1	Antonia	20	Rio	2	3	400
2	Mariana	10	Sao Paulo	2	3	400
3	Juliana	30	Porto Alegre	2	3	400
4	Maria	20	Rio	2	3	400
5	Josiane	30	Curitiba	2	3	400
6	Maria	30	Curitiba	2	3	400
1	Antonia	20	Rio	2	5	100
2	Mariana	10	Sao Paulo	2	5	100
...						

# Exemplo - Cláusula FROM

- Se houver cláusula WHERE, o resultado é o produto cartesiano das tabelas que aparecem na cláusula FROM e a seleção das linhas de acordo com a condição na cláusula WHERE

```
SELECT *  
FROM PECA, FORNECIMENTO  
WHERE PECA.P = FORNECIMENTO.P  
      AND PECA.P = 1
```

P	PNOME	COR	PESO	CIDADE	F	P	QUANTIDADE
1	Parafuso	Vermelho	12.00	Sao Paulo	1	1	200
1	Parafuso	Vermelho	12.00	Sao Paulo	5	1	100

# SQL - Renomeação

- É possível renomear tabelas e colunas para simplificar expressões e evitar conflitos

```
SELECT P.P AS PCOD, P.NOME AS NOME, F.F AS FCOD
FROM PECA P, FORNECIMENTO F
WHERE P.P = F.P
      AND P.P = 1
```

PCOD	NOME	FCOD
1	Parafuso	1
1	Parafuso	5

# Exercício

```
SELECT PACIENTE.NOME, HORA, DATA, MEDICO.NOME
FROM PACIENTE, CONSULTA, MEDICO
WHERE PACIENTE.COD_P = CONSULTA.COD_P AND
      MEDICO.COD_M = CONSULTA.COD_M AND
      MEDICO.COD_M = 7
```

1) Qual o significado da consulta acima?

Obter o nome do paciente, hora e data de cada consulta do médico 7, e o nome do médico.

# Exercício

```
SELECT PACIENTE.NOME, HORA, DATA, MEDICO.NOME
FROM PACIENTE, CONSULTA, MEDICO
WHERE PACIENTE.COD_P = CONSULTA.COD_P AND
      MEDICO.COD_M = CONSULTA.COD_M AND
      MEDICO.COD_M = 7
```

NOME	HORA	DATA	NOME
===== Cleusa	===== 19:30	===== 2000-06-10	===== Mariana
Fernanda	10:00	2000-06-12	Mariana

# Exercícios

- Obter a data da consulta do paciente Caio.
- Obter os nomes dos pacientes do médico Pedrinho.

# SQL - JOIN

- A junção retorna a combinação de colunas de duas tabelas  $T_1$  e  $T_2$  que satisfazem uma condição de junção
- Consultas envolvendo junção

```
SELECT <lista_de_colunas>  
FROM tabela1 join tabela2 on condição_junção  
[join tabela3 on ...]  
[WHERE <condição>]
```

# Exemplo - JOIN

```
SELECT *  
  
FROM PECA JOIN FORNECIMENTO ON PECA.P = FORNECIMENTO.P  
  
WHERE PECA.P = 1
```

P	PNOME	COR	PESO	CIDADE	F	P	QUANTIDADE
1	Parafuso	Vermelho	12.00	Sao Paulo	1	1	200
1	Parafuso	Vermelho	12.00	Sao Paulo	5	1	100

# Exercícios

Faça as seguintes consultas usando junção:

- Obter os nomes dos médicos que tem consulta marcada e as datas das suas consultas
- Obter os nomes dos médicos infectologistas e o andar do ambulatório em que eles atendem.
- Obter os nomes dos pacientes que tem consulta marcada no ambulatório 2.

# SQL - Outer Join

- Junção na qual as linhas de uma ou ambas as tabelas que não são combinadas são mesmo assim preservadas no resultado
- Três tipos

junção externa à esquerda (*left [outer] join*)

- tuplas da relação à esquerda são preservadas

junção externa à direita (*right [outer] join*)

- tuplas da relação à direita são preservadas

junção externa completa (*full [outer] join*)

- tuplas de ambas as relações são preservadas

# Exemplo – LEFT JOIN

```
SELECT *  
FROM PECA LEFT JOIN FORNECIMENTO ON PECA.P = FORNECIMENTO.P  
WHERE CIDADE = 'Sao Paulo'
```

P	PNOME	COR	PESO	CIDADE	F	P	QUANTIDADE
1	'Parafuso'	'Vermelho'	12.00	'Sao Paulo'	1	1	200
1	'Parafuso'	'Vermelho'	12.00	'Sao Paulo'	5	1	100
4	'Martelo'	'Preto'	14.00	'Sao Paulo'	3	4	500
4	'Martelo'	'Preto'	14.00	'Sao Paulo'	5	4	800
6	'Broca'	'Vermelho'	19.00	'Sao Paulo'	4	6	300
6	'Broca'	'Vermelho'	19.00	'Sao Paulo'	5	6	200
10	'Alicate'	'Preto'	20.00	'Sao Paulo'	null	null	null
20	'Regua'	'Branco'	17.00	'Sao Paulo'	null	null	null

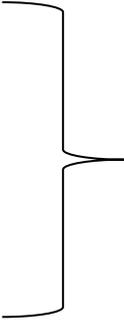
# Exercícios

- Obter o nome de cada médico e respectivos nomes dos pacientes que tem consulta com ele (caso o médico não tenha nenhuma consulta, o nome do paciente deve ser preenchido com null).

# Comandos SQL

➤ Outros comandos, cláusulas e funções:

- UNION / INTERSECT / EXCEPT
- IN / ANY / ALL / EXISTS
- COUNT
- SUM
- AVG
- MAX
- MIN
- GROUP-BY
- ORDER-BY



Funções de Agregação

# Exemplo – Cláusula UNION

```
SELECT CIDADE FROM FORNECEDOR
```

```
UNION
```

```
SELECT CIDADE FROM PECA
```

```
CIDADE
```

```
=====
```

```
Rio
```

```
Sao Paulo
```

```
Porto Alegre
```

```
Curitiba
```

# Exemplo – Cláusula IN

```
SELECT DISTINCT CIDADE FROM FORNECEDOR  
WHERE CIDADE NOT IN (SELECT CIDADE FROM PECA)
```

CIDADE

=====

Curitiba

# Exemplo – Cláusula IN

```
SELECT DISTINCT CIDADE FROM FORNECEDOR  
WHERE CIDADE IN (SELECT CIDADE FROM PECA)
```

CIDADE

=====

Rio

Sao Paulo

Porto Alegre

# SQL - Funções de Agregação

- `COUNT (<lista_de_colunas>)`
- Retorna o número de linhas do resultado de uma consulta

```
SELECT COUNT (*)  
FROM PECA  
  
COUNT  
=====
```

6

```
SELECT COUNT (*) FROM PECA  
WHERE COR = 'Vermelho'  
  
COUNT  
=====
```

2

# SQL - Funções de Agregação

- **SUM (<coluna\_de\_val\_numericos>)**
- Retorna a soma dos valores pertencentes à coluna especificada como argumento

```
SELECT SUM(PESO)  
FROM PECA
```

```
                SUM  
=====
```

91.00
-------

```
SELECT SUM(PESO) FROM PECA  
WHERE CIDADE = 'Rio'
```

```
                SUM  
=====
```

29.00
-------

# SQL - Funções de Agregação

- **AVG (<coluna\_de\_val\_numericos>)**
- Retorna a média dos valores pertencentes à coluna especificada como argumento

```
SELECT AVG(PESO)  
FROM PECA
```

AVG

=====

15.166667

```
SELECT AVG(PESO) FROM PECA  
WHERE CIDADE = 'Rio'
```

AVG

=====

14.500000

# SQL - Funções de Agregação

- **MAX** (<coluna\_de\_val\_numericos>)
- Retorna o maior valor dentre os pertencentes à coluna especificada como argumento

```
SELECT MAX(PESO)  
FROM PECA
```

**MAX**

=====

19.00

```
SELECT MAX(PESO) FROM PECA  
WHERE CIDADE = 'Rio'
```

**MAX**

=====

17.00

# SQL - Funções de Agregação

- **MIN (<coluna\_de\_val\_numericos>)**
- Retorna o menor valor dentre os pertencentes à coluna especificada como argumento

```
SELECT MIN(PESO)  
FROM PECA
```

```
                MIN  
=====
```

12.00
-------

```
SELECT MIN(PESO) FROM PECA  
WHERE CIDADE = 'Rio'
```

```
                MIN  
=====
```

12.00
-------

# SQL - Cláusula GROUP-BY

- Divide o resultado de uma consulta em “grupos” de tuplas cujos valores das colunas especificadas são iguais.

**GROUP BY <lista\_de\_colunas>**

- Exemplo

```
SELECT CIDADE, COUNT(*) FROM PECA
```

```
GROUP BY CIDADE
```

<b>CIDADE</b>	<b>COUNT</b>
=====	=====
Porto Alegre	1
Rio	2
Sao Paul	3

A função COUNT é aplicada separadamente para cada grupo



# SQL - Cláusula GROUP-BY

- Cláusula **HAVING**: especifica uma condição que cada “grupo” deve satisfazer.

- Exemplo

```
SELECT CIDADE, COUNT(*) FROM PECA  
GROUP BY CIDADE HAVING COUNT(*) >= 2
```

CIDADE	COUNT
Rio	2
Sao Paulo	3

A função COUNT é aplicada separadamente para cada grupo



# SQL - Cláusula ORDER-BY

- Determina a ordenação do resultado de uma consulta.

```
SELECT * FROM PECA  
ORDER BY PNOME, CIDADE
```

Ordenação *default* é  
ascendente (ASC)



P	PNOME	COR	PESO	CIDADE
6	Broca	Vermelho	19	Sao Paulo
4	Martelo	Preto	14	Sao Paulo
3	Martelo	Azul	17	Rio
1	Parafuso	Vermelho	12	Sao Paulo
2	Porca	Verde	17	Porto Alegre
5	Prego	Azul	12	Rio

# SQL - Cláusula ORDER-BY

- Determina a ordenação do resultado de uma consulta.

```
SELECT * FROM PECA
```

```
ORDER BY PNOME ASC, CIDADE DESC
```

Ordem  
descendente (DESC)



P	PNOME	COR	PESO	CIDADE
2	Bolt	Verde	17	Paris
5	Cam	Azul	12	Paris
6	Cog	Vermelho	19	Londres
1	Nut	Vermelho	12	Londres
3	Screw	Azul	17	Roma
4	Screw	Vermelho	14	Londres