



UNINASSAU

Banco de Dados

Slides da Disciplina

Prof. Henrique Mota

profhenriquemota@gmail.com

- Modelo Relacional
- Álgebra Relacional
- SQL
- Indexação
- Processamento de Transações
- Protocolos para o Controle de Concorrência
- Processamento de Consultas
 - Otimização de Consultas

Modelo Relacional

- Conceitos Básicos -

- Proposto por Edgar Codd
 - Funcionário da IBM
 - Em 1970
- Baseado em
 - Teoria dos conjuntos
 - Lógica de predicados
- Forte base matemática

Modelo Relacional

- Conceitos Básicos -

- Modelo de dados mais popular do mercado
 - Simplicidade de representação
 - Alto desempenho
 - Garantia de consistência
- Exemplos
 - Oracle
 - SQL Server
 - DB2
 - PostgreSQL
 - MySQL
 - Sybase

Modelo Relacional

- Conceitos Básicos -

- Dados são representados como Relações
 - Relação
 - Sub-conjunto dos produtos cartesianos entre os conjuntos domínio de seus atributos
- Na prática, é utilizado o conceito de Tabela
 - Tabela
 - Dados são organizados em linhas e colunas
 - Cada linha representa um indivíduo
 - Cada coluna representa um atributo dos indivíduos

Modelo Relacional

- Conceitos Básicos -

- Exemplo de Tabela
 - Tabela de Funcionários

<u>matricula</u>	nome	cpf	endereco	salario
1	João	111.111.111-11	Rua ABC, 123	1500,00
2	José	222.222.222-22	Av. XYZ, 456	2500,00
3	Maria	333.333.333-33	Rua DEF, 222	2000,00
4	Pedro	444.444.444-44	Av. JKL, 555	3000,00
5	Paulo	555.555.555-55	Rua GHI, 789	1500,00

- Operadores
 - Álgebra Relacional
 - Cálculo Relacional

- Permitem que os dados das relações sejam manipulados

- São os meios “oficiais” de manipulação de dados no Modelo Relacional

Modelo Relacional

- Conceitos Básicos -

- Linguagens de consulta
 - Permitem manipulação facilitada dos dados de um BD relacional
 - Na prática, são o padrão para manipulação de dados relacionais
 - SQL \Rightarrow Padrão do mercado
 - Geralmente declarativas
 - Você diz *o que* quer, mas não *como* quer que seja executado
 - Comumente convertidas para álgebra relacional durante a execução

Modelo Relacional

- Restrições de Integridade -

- Conjunto de regras que, se *aplicadas*, garantem consistência ao BD
- Mais populares
 - Restrição de Domínio
 - Restrição de Chave
 - Restrição de Integridade Referencial
 - Regras de Negócio

Modelo Relacional

- Restrições de Integridade -

□ Restrição de Domínio

- Um atributo possui um conjunto domínio relacionado
 - A matrícula de um funcionário, por exemplo, possui como conjunto domínio o conjunto dos números inteiros
- A Restrição de Domínio diz que, necessariamente, o valor de um atributo faz parte do conjunto domínio do mesmo
 - Desta forma, torna-se impossível atribuir um valor diferente de um número inteiro a uma matrícula
 - Garantia de integridade!

□ Restrição de Domínio

□ Na prática, a Restrição de Domínio é implementada através de

□ Indicação de tipos de dados

□ Regras de checagem (verificação)

□ Checks

□ Permissão (ou não) de valores nulos

Modelo Relacional

- Restrições de Integridade -

□ Restrição de Chave

- Existe um sub-conjunto diferente de vazio do conjunto total de atributos que *identifica* cada entidade da relação
 - Valores não se repetem!
- A Restrição de Chave garante que um indivíduo do mundo real não apareça mais de uma vez na mesma tabela
 - Garantia de integridade!
- Uma tabela pode ter várias chaves, mas apenas uma delas será a **chave primária**
 - As outras serão **chaves candidatas**
- Cada chave pode ser
 - Simples → Um atributo
 - Composta → Mais de um atributo

Modelo Relacional

- Restrições de Integridade -

□ Restrição de Chave

□ Segundo o Modelo Relacional, toda relação precisa ter ao menos uma chave

□ Teoria dos Conjuntos

□ Um conjunto não pode possuir elementos repetidos

□ Para garantir isso, uma chave deve existir!

□ Na prática, porém, podemos ter tabelas com linhas repetidas

□ É importante que toda tabela possua uma chave!

□ Caso não seja possível encontrar um conjunto de atributos para compor a chave, pode-se incluir um novo atributo com essa finalidade

□ Chave externa

□ Geralmente um código de auto-numeração

Modelo Relacional

- Restrições de Integridade -

- Restrição de Integridade Referencial
 - Uma **chave estrangeira** é um conjunto de campos de uma tabela que é chave primária em outra tabela
 - Utilizada para representar relacionamentos
 - Chave estrangeira funciona como um ponteiro para a outra tabela
 - A Restrição de Integridade Referencial diz que os possíveis valores que chave estrangeira pode assumir em sua tabela estão limitados aos valores da chave primária relacionada a ela na outra tabela
 - Garantia de integridade!

Modelo Relacional

- Restrições de Integridade -

□ Restrição de Integridade Referencial

□ Exemplo

Funcionários

<u>matricula</u>	nome	lotacao
1	João	2
2	José	1
3	Maria	2
4	Pedro	3

Departamentos

<u>codigo</u>	nome
1	Gerência
2	Financeiro
3	Vendas

- Como a coluna “lotacao” é chave estrangeira de Departamentos em Funcionários, os valores dela estão limitados aos valores da chave primária de Departamentos: a coluna “codigo”

Modelo Relacional

- Restrições de Integridade -

□ Regras de Negócio

- Quando as restrições de dados são demasiadamente complexas, é comum que elas sejam colocadas na aplicação
 - Pois muitas vezes o SGBD nem mesmo tem como implementá-las
 - Ainda que o SGBD possa implementá-las, sua aplicação pode interferir no desempenho do restante do sistema
 - SGBD é otimizado para recuperar dados!

Modelo Relacional

- Restrições de Integridade -

□ Regras de Negócio

□ Exemplo

- Um cliente só poderá contrair um empréstimo se possuir salário acima de R\$3.000,00, não possuir restrições de crédito nos últimos cinco anos, não possuir outros empréstimos em outras instituições bancárias, for casado e não possuir mais que três filhos
 - Como implementar eficientemente num BD?

Álgebra Relacional

- Conceitos Básicos -

- ❑ Operador oficial do Modelo Relacional
 - ❑ Padrão “*de direito*”
 - ❑ Desenvolvida pelo próprio Edgar Codd
- ❑ Funciona como uma linguagem de consulta para bancos de dados relacionais
- ❑ Coleção de operações sobre relações
 - ❑ Cada operador recebe relações como operandos
 - ❑ Fornece uma relação como resultado



- ❑ Linguagem procedimental

Álgebra Relacional

- Exemplo Base -

- Admita a seguinte relação nos exemplos a seguir
 - Relação “Funcionarios”

matricula	nome	salario	lotacao
1	Rômulo	1000	1
2	Alex	2000	2
3	João	2500	2

- Operador unário
 - 1 relação como operando
- Símbolo
 - σ
- Finalidade
 - Filtragem horizontal
 - Filtragem de linhas
- Sintaxe
 - $\sigma_{\theta}R$
 - θ : condição lógica
 - R : relação

□ Exemplo

□ Quais os funcionários que ganham mais de R\$1.000?

□ Resposta em Álgebra Relacional

$\sigma_{\text{salario} > 1000} \text{Funcionarios}$

□ Resultado

matricula	nome	salario	lotacao
2	Alex	2000	2
3	João	2500	2

□ Importante

□ O resultado de uma operação da Álgebra Relacional sempre é uma relação!

- Operador unário
 - 1 relação como operando
- Símbolo
 - π
- Finalidade
 - Filtragem vertical
 - Filtragem de colunas
- Sintaxe
 - $\pi_A R$
 - A: lista de colunas
 - R: relação

□ Exemplo

□ Quais os nomes e salários dos funcionários?

□ Resposta em Álgebra Relacional

□ $\pi_{\text{nome, salario}}$ Funcionarios

□ Resultado

nome	salario
Rômulo	1000
Alex	2000
João	2500

□ Importante

□ O resultado de uma operação da Álgebra Relacional sempre é uma relação!

- Sabendo que o resultado de uma operação sempre é uma relação, podemos combinar operadores!
- Exemplo
 - Quais os nomes e salários dos funcionários que trabalham na lotação 2?

□ Resposta em Álgebra Relacional

$\pi_{\text{nome, salario}}(\sigma_{\text{lotacao}=2}\text{Funcionarios})$

□ Resultado

nome	salario
Alex	2000
João	2500

- Assim como na teoria dos conjuntos, é possível utilizar a operação de **união** entre relações
 - A união entre dois conjuntos é um conjunto com todos os elementos dos conjuntos originais
 - A união entre duas relações é uma relação com todas as tuplas das relações originais
- Operador binário
 - 2 relações como operandos
- Símbolo
 - \cup
- Finalidade
 - Unir as tuplas das duas relações

□ Restrições

- As relações envolvidas devem obrigatoriamente ter a mesma quantidade de colunas
- As colunas de mesma posição em cada tabela devem ter tipos de dados compatíveis entre si

□ Sintaxe

- $R_1 \cup R_2$
 - R_1 e R_2 : relações

Álgebra Relacional

- Exemplo Base -

- Admita as seguintes relações no exemplo a seguir
 - Relação “Funcionarios”

matricula	nome	endereco	salario
1	Rômulo	Rua ABC	1000
2	Alex	Rua DEF	2000

- Relação “Clientes”

codigo	nome	endereco
1	Ana	Rua GHI
2	Maria	Av TUV

□ Exemplo

□ Deseja-se criar uma mala direta para todos os funcionários e clientes da empresa. Como conseguir os nomes e endereços de todos os funcionários e clientes juntos?

□ Solução: usar **união**!

□ Resposta em Álgebra Relacional

$$(\pi_{\text{nome, endereco}} \text{Funcionarios}) \cup (\pi_{\text{nome, endereco}} \text{Clientes})$$

□ Resultado

nome_func	end_func
Rômulo	Rua ABC
Alex	Rua DEF
Ana	Rua GHI
Maria	Av TUV

Álgebra Relacional

- Exemplo Base -

□ Admita as seguintes relações nos exemplos a seguir

□ Relação “Funcionarios”

matricula	nome	salario	lotacao
1	Rômulo	1000	1
2	Alex	2000	2
3	João	2500	2

□ Relação “Departamentos”

codigo	nome_dep
1	Gerência
2	Financeiro
3	Vendas

- Operação vinda diretamente da teoria dos conjuntos
 - Cada linha de uma tabela é combinada com todas as linhas da outra tabela, formando uma linha no resultado contendo todas as colunas das duas tabelas
- Operador binário
 - 2 relações como operandos
- Símbolo
 - \times
- Sintaxe
 - $R_1 \times R_2$
 - R_1 e R_2 : relações

□ Exemplo 1

□ Produto cartesiano entre Funcionarios e Departamentos

Funcionarios \times Departamentos

Álgebra Relacional

- Produto Cartesiano -

□ Exemplo:

□ Resultado

matricula	nome	salario	lotacao	codigo	nome_dep
1	Rômulo	1000	1	1	Gerência
1	Rômulo	1000	1	2	Financeiro
1	Rômulo	1000	1	3	Vendas
2	Alex	2000	2	1	Gerência
2	Alex	2000	2	2	Financeiro
2	Alex	2000	2	3	Vendas
3	João	2500	2	1	Gerência
3	João	2500	2	2	Financeiro
3	João	2500	2	3	Vendas

□ Exemplo 2

$\pi_{\text{nome}}(\sigma_{\text{salario} > 1500} \text{Funcionarios}) \times \pi_{\text{nome_dep}} \text{Departamentos}$

□ Resultado

nome	nome_dep
Alex	Gerência
Alex	Financeiro
Alex	Vendas
João	Gerência
João	Financeiro
João	Vendas

- Geralmente o produto cartesiano traz resultados “estranhos”
 - Nos exemplos anteriores, não faz sentido “combinar” departamentos diferentes da lotação do funcionário
- Um produto cartesiano pode ser combinado com a operação de seleção (filtragem de linhas) para produzir resultados mais refinados
 - Chamamos essa operação de **junção**

- Operador derivado

- Produto cartesiano seguido de uma seleção

- Operador binário

- 2 relações como operandos

- Símbolo

- \bowtie

- Sintaxe

- $R_1 \bowtie_{\theta} R_2 \quad \equiv \quad \sigma_{\theta}(R_1 \times R_2)$

- R_1 e R_2 : relações

- θ : condição lógica

□ Exemplo:

□ Quais os nomes de todos funcionários e os nomes dos departamentos onde eles estão lotados?

□ Resposta em Álgebra Relacional

$\pi_{\text{nome, nome_dep}}(\text{Funcionarios} \bowtie_{\text{lotacao=codigo}} \text{Departamentos})$

□ Resultado

nome	nome_dep
Rômulo	Gerência
Alex	Financeiro
João	Financeiro

- Structured Query Language
- Linguagem de consulta para SGBDs relacionais
 - Padrão “*de facto*”
 - Linguagem declarativa
 - Baseada na álgebra relacional e no cálculo relacional

- Histórico:
 - Desenvolvida pela IBM
 - Início dos anos 70
 - Para o uso com o SGBD **System R**
 - Inicialmente chamada SEQUEL
 - Ao final dos anos 70, IBM e ORACLE utilizavam SQL como linguagem de consulta

- Composta por:
 - DML (Data Manipulation Language)
 - Consultas e atualizações → Dados
 - DDL (Data Definition Language)
 - Alterações no esquema do BD → Estrutura
 - DCL (Data Control Language)
 - Controle de usuários e permissões → Acesso
 - DTL (Data Transaction Language)
 - Controle de transações → Multiprocessamento

- Permite manipular a estrutura do BD
- Comandos básicos:
 - CREATE
 - Criação de objetos do BD
 - DROP
 - Exclusão de objetos do BD
 - ALTER
 - Alteração de objetos do BD

- Aplicados a **tabelas**:
 - CREATE TABLE
 - Criação de tabelas
 - DROP TABLE
 - Exclusão de tabelas
 - ALTER TABLE
 - Alteração de tabelas

□ Comando de criação de tabelas

□ Sintaxe:

```
CREATE TABLE nome_tabela (  
    coluna1 tipo_de_dados [NULL|NOT NULL],  
    coluna2 tipo_de_dados [NULL|NOT NULL],  
    coluna3 tipo_de_dados [NULL|NOT NULL],  
    ...  
    restrição_de_integridade1,  
    restrição_de_integridade2,  
    ...  
)
```

□ Sintaxe das restrições de integridade:

□ Chave primária

CONSTRAINT nome_restrição PRIMARY KEY(colunas)

□ Chave candidata

CONSTRAINT nome_restrição UNIQUE(colunas)

□ Checagem

CONSTRAINT nome_restrição CHECK(expressão)

□ Chave estrangeira

CONSTRAINT nome_restrição FOREIGN KEY(colunas)
REFERENCES tabela(colunas)

- Valores nulos
 - Quando a coluna não admitir valores nulos
 - Deve-se usar NOT NULL na definição
 - Quando a coluna admitir valores nulos
 - Pode-se usar NULL
 - Ou simplesmente não indicar nada
- Importante
 - Chaves primárias não podem admitir valores nulos

□ Exemplo

□ Criação da tabela de departamentos:

```
CREATE TABLE Departamentos (  
    codigo INT NOT NULL,  
    nome VARCHAR(50) NOT NULL,  
    CONSTRAINT pk_departamentos  
        PRIMARY KEY(codigo)  
)
```

- Exemplo
 - Criação da tabela de funcionários:

```
CREATE TABLE Funcionarios (  
    matricula INT NOT NULL,  
    nome VARCHAR(50) NOT NULL,  
    cpf CHAR(11) NOT NULL,  
    salario FLOAT NOT NULL,  
    lotacao INT,  
    CONSTRAINT pk_funcionarios  
        PRIMARY KEY(matricula),  
    CONSTRAINT un_funcionarios_cpf UNIQUE(cpf),  
    CONSTRAINT ck_funcionarios_salario CHECK(salario >= 0),  
    CONSTRAINT fk_funcionarios_departamentos  
        FOREIGN KEY(lotacao)  
        REFERENCES Departamentos(codigo)  
)
```

SQL

- CREATE TABLE -

- Restrições de integridade podem ser reduzidas
 - Nomes podem ser omitidos
 - O SGBD atribuirá automaticamente os nomes, de acordo com seu padrão
 - Quando a chave primária ou candidata possuir apenas uma coluna, sua indicação pode ficar na própria definição da coluna
 - Apenas quando possuir uma só coluna!
 - Na criação da chave estrangeira, a coluna referenciada pode ser omitida
 - Será, por padrão, a chave primária da tabela referenciada

- Exemplo
 - Criação da tabela de funcionários:

```
CREATE TABLE Funcionarios (  
    matricula INT PRIMARY KEY NOT NULL,  
    nome VARCHAR(50) NOT NULL,  
    cpf CHAR(11) UNIQUE NOT NULL,  
    salario FLOAT NOT NULL,  
    lotacao INT,  
    CHECK(salario >= 0),  
    FOREIGN KEY(lotacao)  
        REFERENCES Departamentos  
)
```


- ❑ Comando de exclusão de tabelas
- ❑ Sintaxe:

`DROP TABLE nome_tabela`

- Exemplo
 - Exclusão da tabela de funcionários:

DROP TABLE Funcionarios

❑ Importante!

- ❑ Todos os registros da tabela de funcionários seriam excluídos, junto com sua estrutura
- ❑ As exclusões seriam registradas no arquivo de LOG do BD
 - ❑ Uma recuperação das linhas poderia ser efetuada depois da exclusão
- ❑ Existe um comando similar, chamado TRUNCATE
 - ❑ Exclui a estrutura da tabela e suas linhas
 - ❑ Mas não registra as exclusões no arquivo de LOG
 - ❑ Mais rápido, mas extremamente perigoso!

❑ Importante!

- ❑ As restrições de integridade devem ser sempre respeitadas!
- ❑ Caso tentássemos excluir a tabela de departamentos com o comando

DROP TABLE Departamentos

- ❑ Antes da exclusão da tabela de funcionários, um erro iria ocorrer
 - ❑ A restrição de chave estrangeira seria desrespeitada!

□ Comando de alteração de tabelas

□ Sintaxe:

```
ALTER TABLE nome_tabela
```

```
ADD [CONSTRAINT] nome_objeto nova_definição
```

```
ALTER TABLE nome_tabela
```

```
DROP|ALTER COLUMN|CONSTRAINT
```

```
nome_objeto [nova_definição]
```

□ Exemplo

- Inclusão da coluna “rg” na tabela de funcionários:

ALTER TABLE Funcionarios

ADD rg VARCHAR(10)

- Alteração do tamanho da coluna “rg” na tabela de funcionários:

ALTER TABLE Funcionarios

ALTER COLUMN rg VARCHAR(50)

- Exclusão da coluna “rg” na tabela de funcionários:

ALTER TABLE Funcionarios

DROP COLUMN rg

□ Exemplo

- Inclusão da restrição de chave candidata na coluna “rg” da tabela de funcionários:

ALTER TABLE Funcionarios

ADD CONSTRAINT un_funcionarios_rg UNIQUE(rg)

ou

ALTER TABLE Funcionarios ADD UNIQUE(rg)

- Alteração na restrição de checagem do salário da tabela de funcionários:

ALTER TABLE Funcionarios

**ALTER CONSTRAINT ck_funcionarios_salario
CHECK(salario >= 500)**

- Exclusão da restrição de chave estrangeira na lotação na tabela de funcionários:

ALTER TABLE Funcionarios

DROP CONSTRAINT fk_funcionarios_departamentos

- ❑ Permite consultas e atualizações aos dados armazenados
- ❑ Comandos básicos:
 - ❑ SELECT
 - ❑ Consultas
 - ❑ INSERT
 - ❑ Inserções
 - ❑ DELETE
 - ❑ Exclusões
 - ❑ UPDATE
 - ❑ Alterações

- Comando de inserção
- Sintaxe 1:
INSERT INTO tabela [(lista_de_colunas)]
VALUES (lista_de_valores)
- Sintaxe 2:
INSERT INTO tabela [(lista_de_colunas)]
SELECT ...

□ Exemplo:

- Adicione o funcionário “João da Silva”, de matrícula 123 e salário R\$2.500,00 à tabela de funcionários
- Consulta em SQL:

**INSERT INTO Funcionarios(matricula, nome,
salario)**

VALUES (123, ‘João da Silva’, 2500)

- Comando de alteração

- Sintaxe:

UPDATE tabela

SET coluna1 = valor1 [, coluna2 = valor2...]

[WHERE condição]

□ Exemplo:

- Os funcionários com salários menores que R\$2.000,00 receberão um aumento de 25%
- Consulta em SQL:

UPDATE Funcionarios

SET salario = salario * 1.25

WHERE salario < 2000

- Comando de remoção

- Sintaxe:

DELETE FROM tabela
[WHERE condição]

□ Exemplo:

- Apague da tabela de funcionários todos os funcionários de salário inferior a R\$1.000,00
- Consulta em SQL:

DELETE FROM Funcionarios
WHERE salario < 1000

- ❑ Comando básico de consulta

- ❑ Sintaxe:

SELECT lista_de_colunas

FROM lista_de_tabelas

[WHERE condição]

[GROUP BY expressão_de_agrupamento]

[HAVING condição]

[ORDER BY expressão_de_ordenamento]

□ Exemplo:

□ Tabela “Funcionarios”

id	nome	salario
1	Rômulo	1000
2	Alex	2000
3	João	2500
4	José	1500

□ Exemplo:

- Consulta que retorne os nomes e salários de todos os funcionários que tenham salários a partir de R\$ 2.000,00.
- Consulta em SQL:

```
SELECT nome, salario  
FROM Funcionarios  
WHERE salario >= 2000
```

□ Exemplo:

□ Resultado:

nome	salario
Alex	2000
João	2500

□ Importante!

□ O resultado de um SELECT é sempre uma tabela!

- Toda consulta em SQL é convertida para álgebra relacional no momento de sua execução

**SELECT nome, salario
FROM Funcionarios
WHERE salario >= 2000**



$\pi_{\text{nome, salario}}(\sigma_{\text{salario} \geq 2000}(\text{Funcionarios}))$

- Cláusula para ordenar os resultados de um SELECT
- A cláusula ORDER BY é opcional
 - Quando existir deve ser a última do SELECT
 - Sempre!

□ Exemplo:

□ Tabela “Funcionarios”

matricula	nome	salario
1	Rômulo	1000
2	Alex	2000
3	João	2500
4	José	1500

□ Exemplo:

- Consulta que retorne os nomes e salários de todos os funcionários ordenados pelo nome do funcionário
- Consulta em SQL:

```
SELECT nome, salario  
FROM Funcionarios  
ORDER BY nome
```

q Exemplo:

↳ Resultado:

nome	salario
Alex	2000
João	2500
José	1500
Rômulo	1000

- A “expressão de ordenação” deve ser uma lista de colunas ou expressões matemáticas envolvendo colunas
 - Quando a ordenação for ascendente, deve-se utilizar a palavra ASC, ou não informar nada
 - Quando a ordenação for descendente, deve-se utilizar a palavra DESC

□ Exemplo:

- Consulta que retorne os nomes dos funcionários ordenados pelos salários de forma decrescente
- Consulta em SQL:

```
SELECT nome  
FROM Funcionarios  
ORDER BY salario DESC
```

□ Exemplo:

□ Resultado:

nome
João
Alex
José
Rômulo

□ Importante!

□ As colunas participantes da cláusula ORDER BY não precisam aparecer nos resultados

- Operação vinda diretamente da teoria dos conjuntos
 - Cada linha de uma tabela é combinada com todas as linhas da outra tabela, formando uma linha no resultado contendo todas as colunas das duas tabelas
- Sintaxe
 - Lista de tabelas separadas por vírgula na cláusula FROM do comando SELECT

□ Exemplo:

□ Tabela “Funcionarios”

matricula	nome	salario	lotacao
1	Rômulo	1000	1
2	Alex	2000	2
3	João	2500	2

□ Tabela “Departamentos”

codigo	nome_dep
1	Gerência
2	Financeiro
3	Vendas

q Exemplo:

```
SELECT *  
FROM Funcionarios, Departamentos
```

□ Exemplo:

□ Resultado

matricula	nome	salario	lotacao	codigo	nome_dep
1	Rômulo	1000	1	1	Gerência
1	Rômulo	1000	1	2	Financeiro
1	Rômulo	1000	1	3	Vendas
2	Alex	2000	2	1	Gerência
2	Alex	2000	2	2	Financeiro
2	Alex	2000	2	3	Vendas
3	João	2500	2	1	Gerência
3	João	2500	2	2	Financeiro
3	João	2500	2	3	Vendas

q Exemplo:

```
SELECT nome, nome_dep  
FROM Funcionarios, Departamentos  
WHERE salario > 1500
```

□ Exemplo:

□ Resultado

nome	nome_dep
Alex	Gerência
Alex	Financeiro
Alex	Vendas
João	Gerência
João	Financeiro
João	Vendas

- Geralmente o produto cartesiano traz resultados “estranhos”
 - Nos exemplos anteriores, não faz sentido “combinar” departamentos diferentes da lotação do funcionário
- Um produto cartesiano pode ser combinado com a cláusula **WHERE** (filtragem de linhas) para produzir resultados mais refinados
 - Chamamos essa operação de **junção**

q Exemplo:

```
SELECT nome, nome_dep  
FROM Funcionarios, Departamentos  
WHERE lotacao = codigo
```

□ Exemplo:

□ Resultado

nome	nome_dep
Rômulo	Gerência
Alex	Financeiro
João	Financeiro

- Existe uma sintaxe específica para junção:

SELECT lista_de_colunas

FROM tabela1

[INNER] JOIN tabela2

ON condição

□ Exemplo:

```
SELECT nome, nome_dep  
FROM Funcionarios, Departamentos  
WHERE lotacao = codigo
```

⇓

```
SELECT nome, nome_dep  
FROM Funcionarios  
      JOIN Departamentos  
      ON lotacao = codigo
```

- As duas sintaxes são equivalentes
 - Retornam exatamente os mesmos resultados

- Em teoria, a sintaxe (2) é mais rápida
 - Quando um SGBD recebe uma consulta na sintaxe (1), ele a converte para a sintaxe (2)
 - Durante a fase de otimização
 - Heurística

- É possível “apelidar” as tabelas envolvidas nas suas consultas
 - Ou seja, dar novos nomes às tabelas, durante a execução da sua consulta
- Bastante útil para
 - Alterar os nomes para facilitar desenvolvimento da consulta
 - Referenciar colunas que originalmente possuem o mesmo nome (ambíguas)
 - Permitir auto-junção

□ Exemplo:

- Retorne o nome do funcionário e o nome do departamento onde ele está lotado
- Consulta em SQL:

```
SELECT f.nome, d.nome  
FROM Funcionarios f,  
Departamentos d  
WHERE codigo = lotacao
```


SQL

- Apelidos de Tabelas -

□ Exemplo:

□ Resultado:

f.nome	d.nome
Rômulo	Gerência
Alex	Financeiro
João	Financeiro

□ Importante!

□ Quais os nomes das colunas do resultado?

- Como as colunas são ambíguas, não há como saber
- Será dependente do SGBD
- Muito provavelmente seria algo próximo de f.nome e d.nome

SQL

- Apelidos de Colunas -

- É possível “apelidar” as colunas dos resultados das suas consultas
 - Ou seja, dar novos nomes a colunas do seu resultado
- Bastante útil para
 - Alterar os nomes de acordo com a necessidade de uso
 - Desambiguar colunas que originalmente possuem o mesmo nome

□ Exemplo:

- Retorne o nome do funcionário e o nome do departamento onde ele está lotado
- Consulta em SQL:

```
SELECT f.nome as nome_fun,  
       d.nome as nome_dep  
FROM Funcionarios f,  
       Departamentos d  
WHERE codigo = lotacao
```

SQL

- Apelidos de Colunas -

□ Exemplo:

□ Resultado:

nome_fun	nome_dep
Rômulo	Gerência
Alex	Financeiro
João	Financeiro

□ Importante!

□ No resultado da consulta, os nomes das colunas passam a ser *nome_fun* e *nome_dep*

□ Garantido!

- No Modelo Relacional, assim como na Teoria dos Conjuntos na qual ele é fundamentado, não há repetição de elementos
 - Mas os SGBDs comerciais permitem linhas repetidas
- O comando DISTINCT, parte da cláusula SELECT, serve para omitir linhas repetidas do resultado das consultas

□ Exemplo:

- Retorne o nome dos departamentos que possuem funcionários lotados
- Se executarmos a consulta em SQL:

```
SELECT d.nome as nome_dep  
FROM Funcionarios f,  
Departamentos d  
WHERE codigo = lotacao
```

□ Exemplo:

□ Obteremos o resultado:

nome_dep
Gerência
Financeiro
Financeiro

□ O departamento “Financeiro” está repetido!

□ Como resolver?

□ Utilizando DISTINCT!

□ Exemplo:

- Se alterarmos a consulta em SQL para:

```
SELECT DISTINCT d.nome as nome_dep  
FROM Funcionarios f, Departamentos d  
WHERE codigo = lotacao
```

- Obteremos o resultado:

nome_dep
Gerência
Financeiro

- Sem repetição de linhas!

- Assim como na teoria dos conjuntos, é possível utilizar a operação de **união** entre relações (tabelas)
 - A união entre dois conjuntos é um conjunto com todos os elementos dos conjuntos originais
 - A união entre duas relações é uma relação com todas as tuplas das relações originais

- Restrições para o uso da união
 - As relações envolvidas devem obrigatoriamente ter a mesma quantidade de colunas
 - As colunas de mesma posição em cada tabela devem ter tipos de dados compatíveis entre si

□ Sintaxe:

```
SELECT ...  
UNION [ALL]  
SELECT ...
```

□ É executada uma união entre o resultado do primeiro SELECT e o resultado do segundo SELECT

□ Tabela “Funcionarios”

matricula	nome_func	end_func	salario
1	Rômulo	Rua ABC	1000
2	Alex	Rua DEF	2000
3	João	Av XYZ	2500
4	Pedro	Rua JKL	1500

□ Tabela “Clientes”

codigo	nome_cliente	end_cliente
1	Ana	Rua GHI
2	Maria	Av TUV
3	Pedro	Rua JKL

- Problema: deseja-se criar uma mala direta para todos os funcionários e clientes da empresa. Como conseguir os nomes e endereços de todos os funcionários e clientes juntos?
 - Solução: usar **união**!

□ Solução:

```
SELECT nome_func, end_func  
FROM Funcionarios
```

UNION

```
SELECT nome_cliente, end_cliente  
FROM Clientes
```

□ Resultado:

nome_func	end_func
Rômulo	Rua ABC
Alex	Rua DEF
João	Av XYZ
Ana	Rua GHI
Maria	Av TUV
Pedro	Rua JKL

□ Importante!

- Os nomes das colunas no resultado serão os nomes das colunas da primeira relação
 - Apelidos para as colunas podem ser utilizados para que os nomes das colunas fiquem mais adequados

□ Solução com apelidos:

```
SELECT nome_func as nome,  
       end_func as endereco  
FROM Funcionarios  
UNION  
SELECT nome_cliente, end_cliente  
FROM Clientes
```


q Resultado:

nome	endereco
Rômulo	Rua ABC
Alex	Rua DEF
João	Av XYZ
Ana	Rua GHI
Maria	Av TUV
Pedro	Rua JKL

□ Curiosidade

- A operação de união remove tuplas repetidas do resultado
 - Caso não tenha percebido, “Pedro” é funcionário e cliente ao mesmo tempo
 - Ele foi incluído apenas uma vez no resultado do UNION!
- Caso seja importante manter as tuplas repetidas, deve ser utilizado o comando **UNION ALL**

- Solução com UNION ALL:

```
SELECT nome_func as nome,  
       end_func as endereco  
FROM Funcionarios
```

UNION ALL

```
SELECT nome_cliente, end_cliente  
FROM Clientes
```

□ Resultado:

nome	endereco
Rômulo	Rua ABC
Alex	Rua DEF
João	Av XYZ
Pedro	Rua JKL
Ana	Rua GHI
Maria	Av TUV
Pedro	Rua JKL

- Em algumas situações, será importante preservar as duplicatas; em outras, será importante que elas sejam omitidas
 - Use UNION ou UNION ALL adequadamente!

- Na teoria do Modelo Relacional, valores nulos não existem!
 - Todo e qualquer atributo deve conter um valor

- Mas os SGBDs relacionais permitem que atributos recebam valores nulos (desde que permitidos na especificação das colunas)

- ❑ Interpretação errônea sobre o valor nulo:
 - ❑ Valor não existe

- ❑ Significado correto de um valor nulo
 - ❑ Valor existe, mas não se sabe no momento

- ❑ Uma coisa é existir, mas não se saber; outra coisa é não existir!

□ Tabela “Funcionarios”

matricula	nome	endereco	salario	data_nascimento
1	Rômulo	Rua ABC	1000,00	04/10/1970
2	Alex	Rua DEF	1500,00	06/09/1980
3	João	Av XYZ	3000,00	NULO
4	Ana	Rua GHI	2000,00	01/01/2000
5	Pedro	Av TUV	2500,00	NULO

□ Exemplo:

- Quais os nomes dos funcionários que não possuem data de nascimento informada?

```
SELECT nome  
FROM Funcionarios  
WHERE data_nascimento IS NULL
```


□ Resultado:

nome
João
Pedro

□ Exemplo:

- Quais os nomes dos funcionários que possuem data de nascimento informada?

```
SELECT nome  
FROM Funcionarios  
WHERE data_nascimento IS NOT NULL
```

□ Resultado:

nome
Rômulo
Alex
Ana

❑ Importante

- ❑ Toda e qualquer comparação a valores nulos que não seja feita com o operador IS (ou IS NOT) retornará falso!

❑ A consulta

SELECT nome

FROM Funcionarios

WHERE data_nascimento = NULL

- ❑ Retornará uma tabela vazia!

- ❑ Por quê?

- ❑ O operador ternário BETWEEN, bastante utilizado na prática, compara se um valor está compreendido dentro de um intervalo informado
- ❑ Sintaxe:
valor BETWEEN inicio AND fim

□ Exemplo:

□ Qual o nome dos funcionários que ganham salário entre R\$1.500,00 e R\$2.500,00?

□ Solução:

SELECT nome

FROM Funcionarios

WHERE **salario BETWEEN 1500 AND 2500**

- A consulta do exemplo anterior é equivalente a
SELECT nome
FROM Funcionarios
WHERE salario >= 1500 AND salario <= 2500
- Os pontos extremos fazem parte do intervalo!

- ❑ Se o problema fosse:
 - ❑ Qual o nome dos funcionários que ganham salário maior que R\$1.500,00 e menor que R\$2.500,00?
- ❑ Solução:
SELECT nome
FROM Funcionarios
WHERE **salario > 1500 AND salario < 2500**
- ❑ BETWEEN não poderia ser utilizado!
 - ❑ Pontos extremos não fazem parte do intervalo

- ❑ Operação de Agregação

- ❑ Reduzir um conjunto de dados a um valor que represente o conjunto inteiro

- ❑ De acordo com o efeito que se deseja

- ❑ Por exemplo:

- ❑ Notas: 10, 9.5, 8, 7.5, 9

- ❑ Qual a **média** das notas?

- ❑ Média = $(10 + 9.5 + 8 + 7.5 + 9) / 5 = 8.8$

- ❑ A média das notas é uma agregação!

- ❑ Conjunto original de 5 valores foi reduzido a apenas um

- Funções de agregação mais comuns
 - SUM: soma
 - COUNT: contagem
 - AVG: média
 - MAX: máximo
 - MIN: mínimo

- Algumas funções de agregação podem ser exclusivas de cada SGBD

□ Exemplo:

- Qual o valor da folha de pagamento da empresa (soma dos salários de todos os funcionários)?

```
SELECT SUM(salario)
FROM Funcionarios
```

- Qual a média salarial dos funcionários?

```
SELECT AVG(salario)
FROM Funcionarios
```

□ Exemplo:

- Qual o maior de todos os salários?

```
SELECT MAX(salario)  
FROM Funcionarios
```

- Qual o menor de todos os salários?

```
SELECT MIN(salario)  
FROM Funcionarios
```

□ Exemplo:

- Qual a quantidade total de funcionários?

```
SELECT COUNT(*)  
FROM Funcionarios
```

- Qual a quantidade de funcionários que ganham mais de R\$2.000,00?

```
SELECT COUNT(*)  
FROM Funcionarios  
WHERE salario > 2000
```

- É comum o uso de apelidos
 - Nomes mais adequados e independentes do SGBD
- Exemplo:
 - Qual a folha de pagamento total e média salarial dos funcionários?

```
SELECT SUM(salario) AS folha_pgto,  
        AVG(salario) AS media_salarial  
FROM Funcionarios
```

- A operação de agrupamento cria grupos de linhas
 - Uma vez que os grupos estão criados, operações específicas podem ser aplicadas a eles
- Exemplo:
 - Qual a média salarial dos funcionários por lotação?

```
SELECT lotacao, AVG(salario)  
FROM Funcionarios  
GROUP BY lotacao
```
 - Qual o maior e o menor salário de cada lotacao?

```
SELECT lotacao, MAX(salario), MIN(salario)  
FROM Funcionarios  
GROUP BY lotacao
```

□ Exemplo:

- Quais as lotações com média salarial dos funcionários maior que R\$2.000,00?

SELECT lotacao

FROM Funcionarios

GROUP BY lotacao

HAVING AVG(salario) > 2000

- A cláusula HAVING efetua filtrações dentro de cada grupo criado
- Filtrações no HAVING podem conter funções de agregação

□ Exemplo:

- Quais os nomes dos departamentos com média salarial dos funcionários maior que R\$2.000,00?

```
SELECT D.nome  
FROM Funcionarios F  
JOIN Departamentos D ON F.lotacao = D.codigo  
GROUP BY D.nome  
HAVING AVG(F.salario) > 2000
```

- Agrupamento pode ser utilizado em conjunto com as outras operações do SELECT (como a junção)

- Algumas consultas precisam buscar valores já presentes no BD para utilizá-los em suas condições de comparação
 - Assim, podemos inserir consultas SQL dentro de outras consultas SQL
 - Sub-consultas
 - Consultas aninhadas

- Operadores utilizados com sub-consultas
 - IN: testa se o valor a ser comparada está dentro dos resultados da sub-consulta
 - {=|<>|>|<|>=|<=} ANY: testa se a operação desejada é verdadeira para qualquer dos resultados da sub-consulta
 - {=|<>|>|<|>=|<=} ALL: testa se a operação desejada é verdadeira para todos os resultados da sub-consulta
 - EXISTS: testa se a sub-consulta possui resultados

- Os operadores possuem suas formas negativas
 - NOT IN
 - NOT EXISTS

□ Exemplos

- Qual o nome do funcionário de maior salário?

```
SELECT nome  
FROM Funcionarios  
WHERE salario IN (SELECT max(salario) FROM Funcionarios)
```

- Qual o nome do funcionário de segundo maior salário?

```
SELECT nome  
FROM Funcionarios  
WHERE salario IN  
    (SELECT max(salario)  
    FROM Funcionarios  
    WHERE salario < (SELECT max(salario) FROM Funcionarios))
```

- ❑ Se a sub-consulta tiver comparações de valores com dados da consulta externa
 - ❑ Ela será uma sub-consulta **correlacionada**
- ❑ Problema: o tempo de processamento tende a ser alto
 - ❑ Uma sub-consulta não-correlacionada pode ser executada uma única vez e ter seus valores avaliados pela consulta externa tantas vezes quanto forem necessárias
 - ❑ Uma sub-consulta correlacionada deverá ser executada para cada avaliação necessária na consulta externa
 - ❑ Ou seja, será executada várias vezes durante a execução da consulta externa
 - ❑ Sub-consultas correlacionadas devem ser usadas com parcimônia

□ Exemplo

□ Tabela “Funcionarios”

matricula	nome	salario	lotacao
1	Rômulo	1000	1
2	Alex	2000	2
3	João	2500	2

□ Tabela “Projetos”

codigo	nome_pro
1	Projeto X
2	Projeto Y

Tabela “Func_Proj”

matricula	codigo
1	1
1	2
2	2

□ Exemplos

- Quais o nomes dos funcionários que estão trabalhando em algum projeto?

```
SELECT nome  
FROM Funcionarios F  
WHERE EXISTS (SELECT matricula FROM Func_Proj  
              WHERE matricula = F.matricula)
```

- A consulta acima poderia ser reescrita como

```
SELECT nome  
FROM Funcionarios  
WHERE matricula IN (SELECT matricula FROM Func_Proj)
```

□ Exemplos

- Outra forma de resolver a consulta de funcionário de maior salário:

SELECT nome

FROM Funcionarios F

WHERE salario > ALL (SELECT salario FROM Funcionarios
WHERE matricula <> F.matricula)

- Nem sempre será possível evitar sub-consultas correlacionadas
 - Quando for, evite-as!

- Os SGBD relacionais suportam um tipo de objeto chamado **visão**
 - Tabelas virtuais
 - Baseadas em consultas SQL
 - É comparável a promover uma consulta SQL ao nível de tabela

- Exemplo
 - Suponha que a seguinte consulta SQL seja demasiadamente utilizada

```
SELECT nome, nome_dep
FROM Funcionarios
JOIN Departamentos ON lotacao=codigo
```

□ Exemplo

- Podemos criar uma visão chamada Func_Dep_View, com o comando

```
CREATE VIEW Func_Dep_View AS  
  SELECT nome, nome_dep  
  FROM Funcionarios  
  JOIN Departamentos ON lotacao=codigo
```

- Assim, sempre que o resultado da consulta anterior for necessário, bastará se executar a consulta

```
SELECT * FROM Func_Dep_View
```

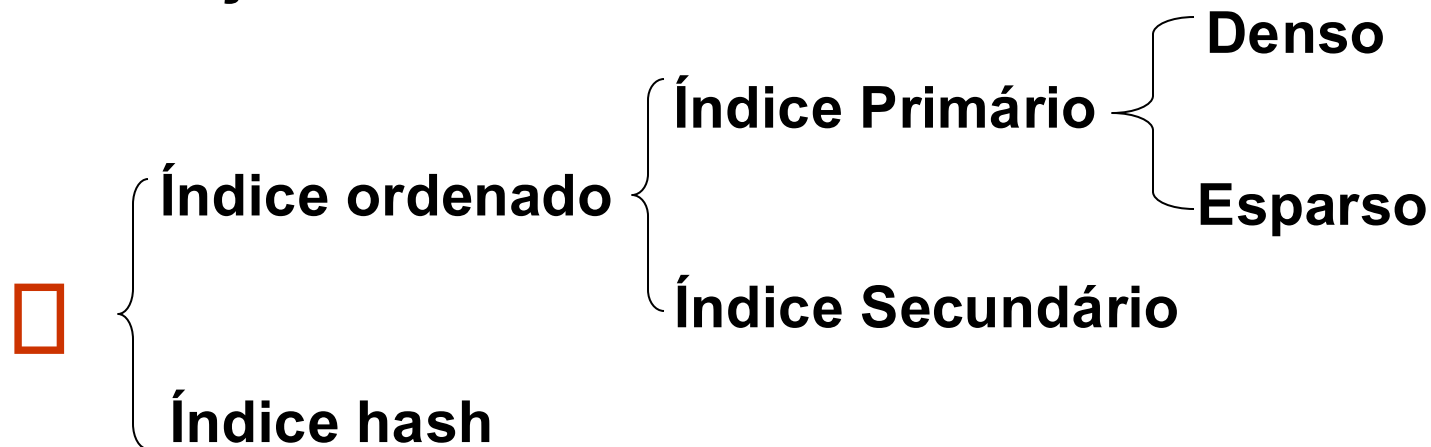
- A visão funciona como uma tabela (virtual)!

□ Vantagens

- O usuário final verá apenas o resultado da visão, não precisando saber os detalhes de como estão construídas as tabelas ou de como foi especificada a consulta
 - Encapsulamento
 - Desacoplamento
 - Melhor manutenibilidade
- Visões podem receber permissões de acesso personalizadas
 - Segurança

- Índices
 - Cada estrutura de índice está associada a uma chave de busca
 - Chave de busca representa um atributo de tuplas
 - Fornecem um caminho através do qual os dados podem ser localizados e acessados de forma mais eficiente
 - Reduzem o overhead de busca a um pequeno conjunto de tuplas

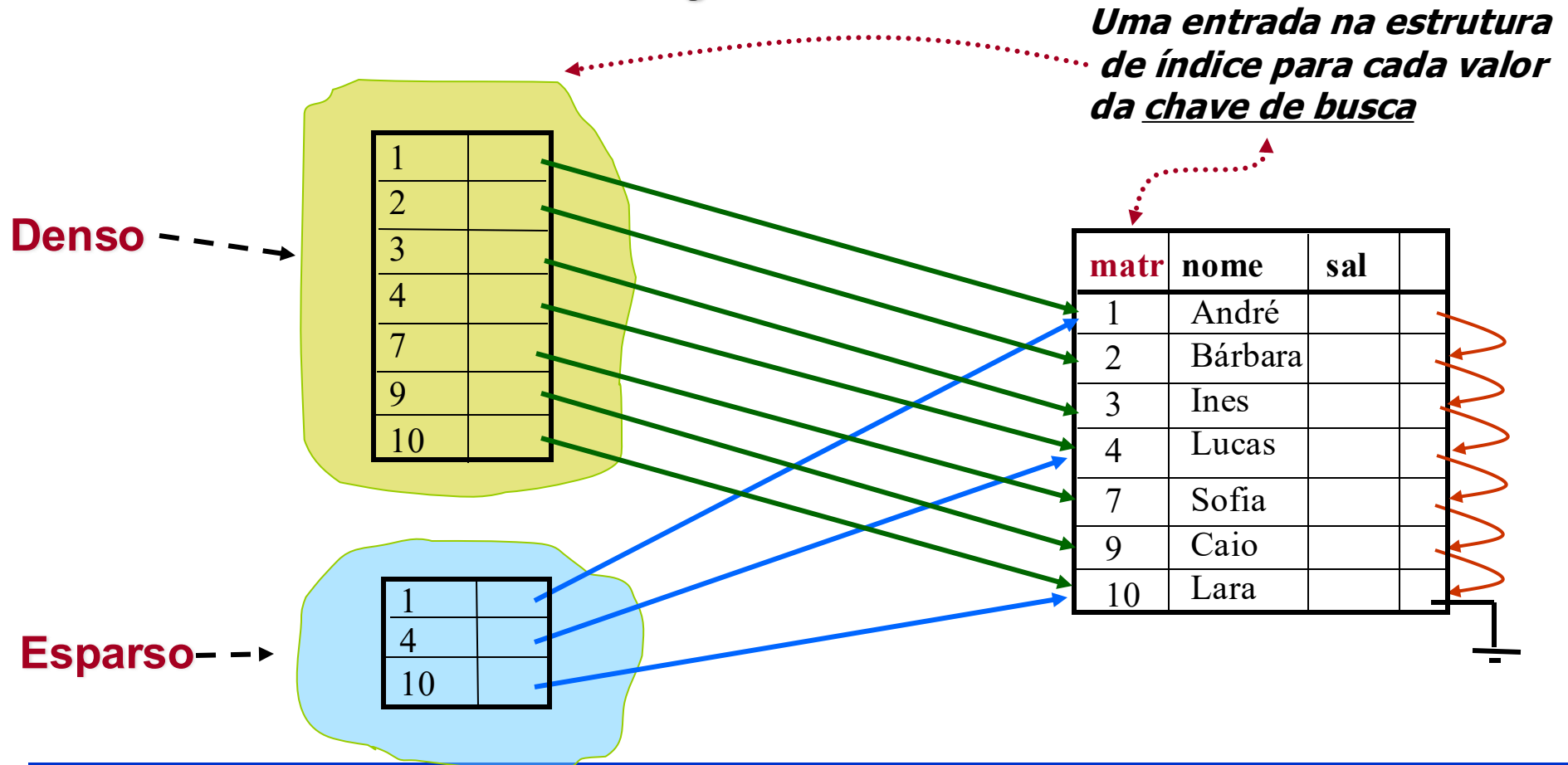
□ Classificação de índices



Indexação

- Tipos de Índices -

- Índice Primário (*Clustering Indices*)
 - Definido sobre uma chave de busca que determina a ordem física dos registros

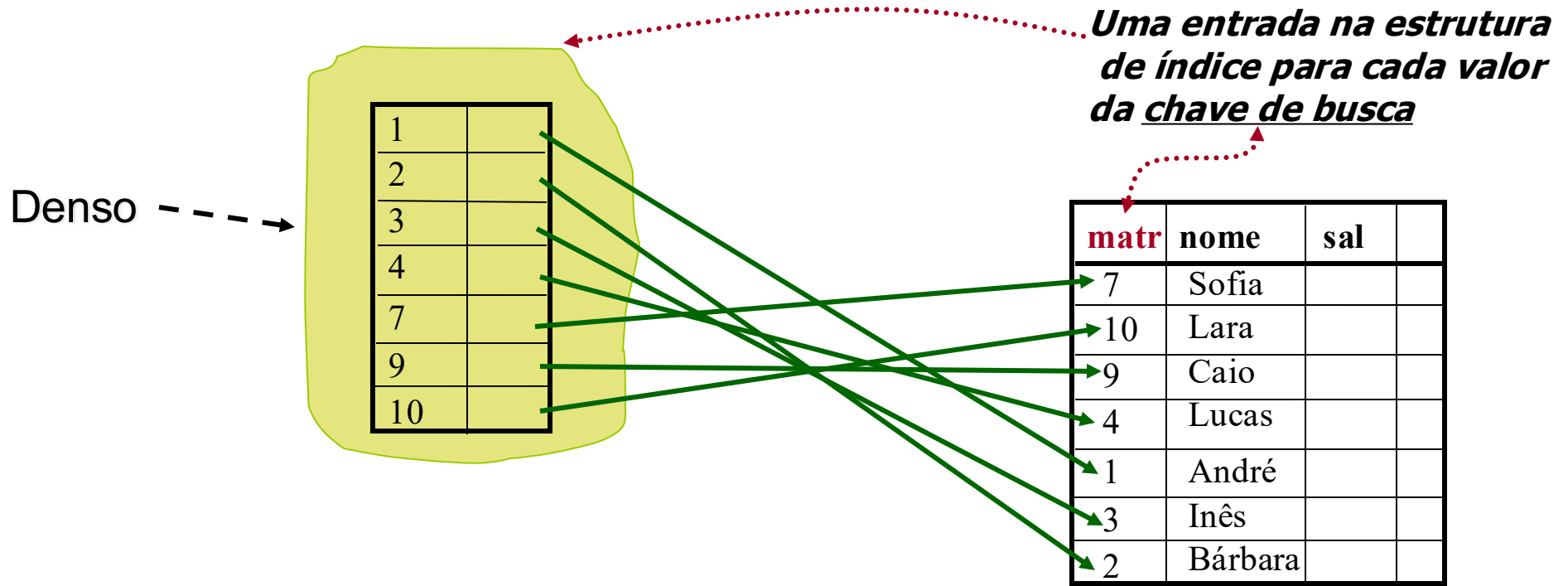


Indexação

- Tipos de Índices -

□ Índice Secundário

- Chave de busca não determina a ordem física das tuplas
 - Índices determinam uma ordem (lógica) das tuplas
- Exemplo
 - Chave de busca definida sobre atributo chave

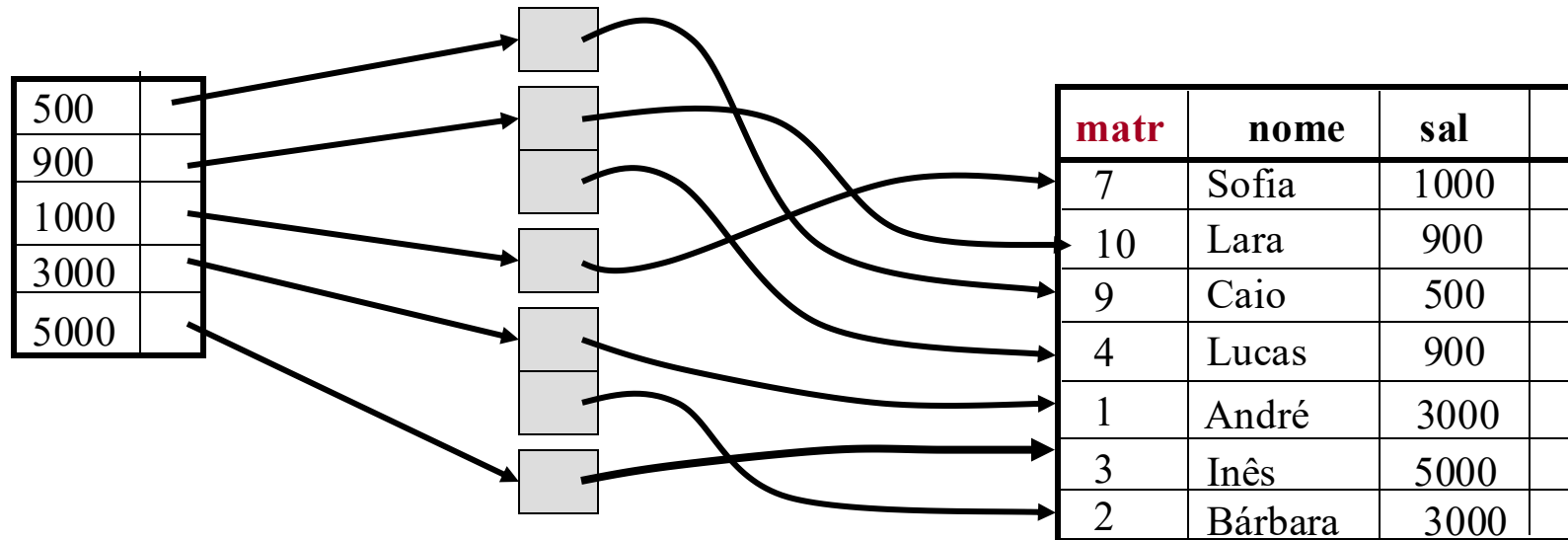


□ Índice secundário é sempre denso

Indexação

- Tipos de Índices -

- Índice Secundário (cont.)
 - Chave de busca definida sobre atributo não chave
 - Nível extra de indireção



□ Índice Hash

- Estrutura de índice organizada como um arquivo hash

□ Hashing

- **Bucket** representa uma unidade armazenamento de um conjunto de registros

- Função de Hash

$$h: K \mapsto B, \text{ onde}$$

K representa o conjunto de todas as chaves de busca e
 B é o conjunto do endereço de todos os buckets

- Eficiente para consultas do tipo *exact match*

- Utilizado para implementar *select* com igualdade sobre atributos chaves

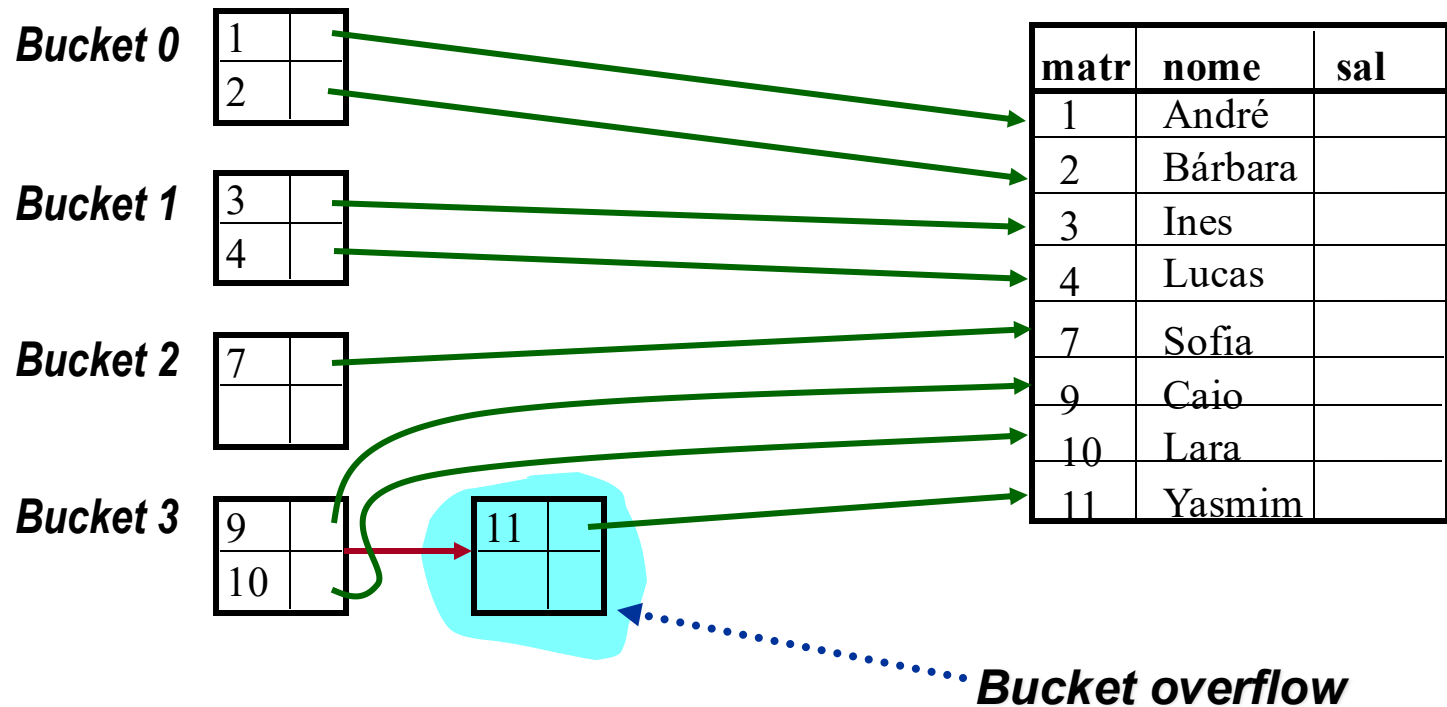
Select $A_1, A_2, A_3, \dots, A_n$
From s
where $A_k = c$

Indexação

- Tipos de Índices -

- Índice Hash (cont.)
 - Hashing (cont.)

$$h(matr) = \lfloor matr / 3 \rfloor$$



- Estimativa de custo (sem overflow)
 - 2 acessos a disco

□ Índice *Hash* (cont.)

□ **Hashing** (cont.)

□ Fatores geradores de *overflow* de *buckets*

□ Número insuficiente de buckets

□ Altas taxas de colisão

□ Número ideal de buckets n_B é dado por

$n_B > n_R / f_R$, onde

+ n_R representa o número de tuplas e

+ f_R representa o número de tuplas por *bucket*

□ Desequilíbrio

□ Poucos buckets armazenam mais tuplas que os outros buckets

+ *Overflow* de *buckets*, mesmo com *buckets* com espaço

□ Várias tuplas com mesmo valor de chave de busca

□ Função hash não garante uma distribuição uniforme

- Índice *Hash* (cont.)

- **Hashing** (cont.)

- Função de hash é escolhida no momento em que a estrutura de índice está sendo criado
 - Função de *hash*
 - Mapea um valor de chave de busca em um endereço pertencente ao conjunto **B**
 - **B** muito grande
 - Desperdício de espaço em disco
 - **B** muito pequeno
 - *Overflow* de *buckets*
 - Alteração da função de *hash*
 - Reorganização

□ Árvores **B** (*B-trees*)

□ Árvore balanceada

□ Todos os caminhos a partir da raiz até os nós folhas apresentam o mesmo comprimento

□ Altura da árvore é constante

□ Regras de formação de uma *B-Tree* de ordem n

□ Cada nó interno de uma *B-tree* é da forma

$\langle P_1, \langle K_1, PD_1 \rangle, P_2, \langle K_2, PD_2 \rangle, \dots, P_{m-1}, \langle K_{m-1}, PD_{m-1} \rangle, P_m \rangle$

□ $m \leq n$

□ Cada P_i representa um ponteiro para uma subárvore (*B-tree*)

□ ponteiro de árvore

□ Cada PD_i representa um ponteiro para a página que contém uma tupla r cuja chave de busca é igual a K_i

□ Ponteiro de dados

- Árvores **B** (cont.)
 - Regras de formação de uma *B-Tree* de ordem ***n*** (cont.)
 - Dentro de um nó: $K_1 < K_2 < \dots < K_{m-1}$
 - Para todo valor X de chave de busca na subárvore apontada por P_i , temos
$$K_{i-1} < X < K_i \text{ para } 1 < i < m, X < K_i \text{ para } i=1 \text{ e } K_{i-1} < X \text{ para } i=m$$
 - Cada nó possui no máximo ***n*** ponteiros de árvore
 - Cada nó possui no mínimo $\lceil n/2 \rceil$ ponteiros de árvore
 - $\lfloor (n-1)/2 \rfloor$ valores de chave de busca
 - Todos os nós exceto a raiz
 - A raiz possui no mínimo dois ponteiros de árvore
 - Um nó com ***m*** ponteiros de árvore possui ***m-1*** valores de chave de busca e ***m-1*** ponteiros de dados

□ Árvores **B** (cont.)

□ Regras de formação de uma *B-Tree* de ordem ***n*** (cont.)

□ Todos os nós folhas possuem a altura e apresentam a mesma estrutura que os nós internos

□ Os ponteiros de árvore têm valor ***null***

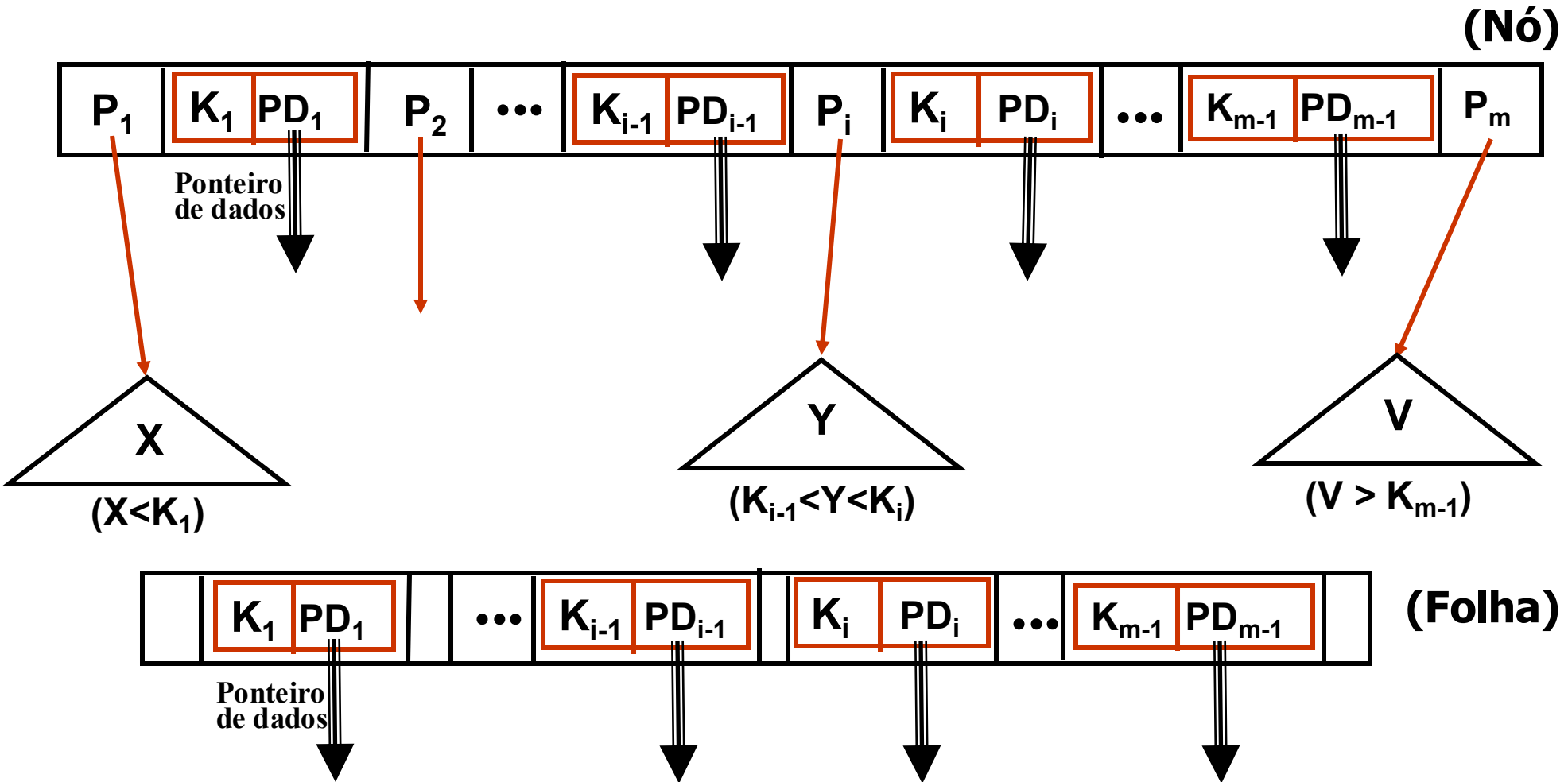
□ Quantidade máxima de valores de chave de busca que podem ser armazenados na raiz de uma *B-tree* de ordem p

$p-1$

Indexação

- Estruturas de Índices Ordenados -

- Árvores **B** (cont.)
- Estrutura de uma árvore **B**



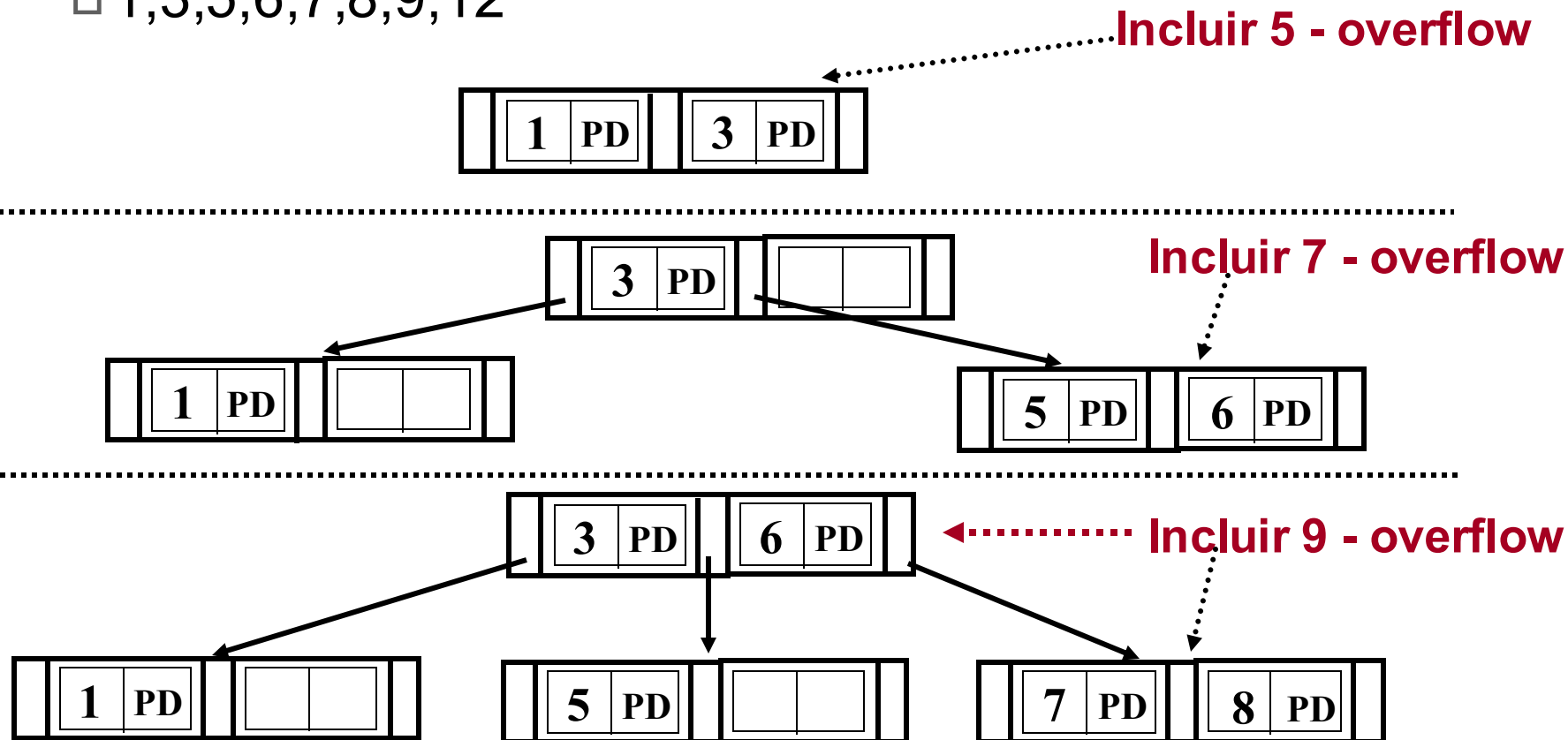
Indexação

- Estruturas de Índices Ordenados -

□ Árvores B (cont.)

□ Construa árvore B de ordem 3 com a seguinte ordem de inserção de valores de chave de busca

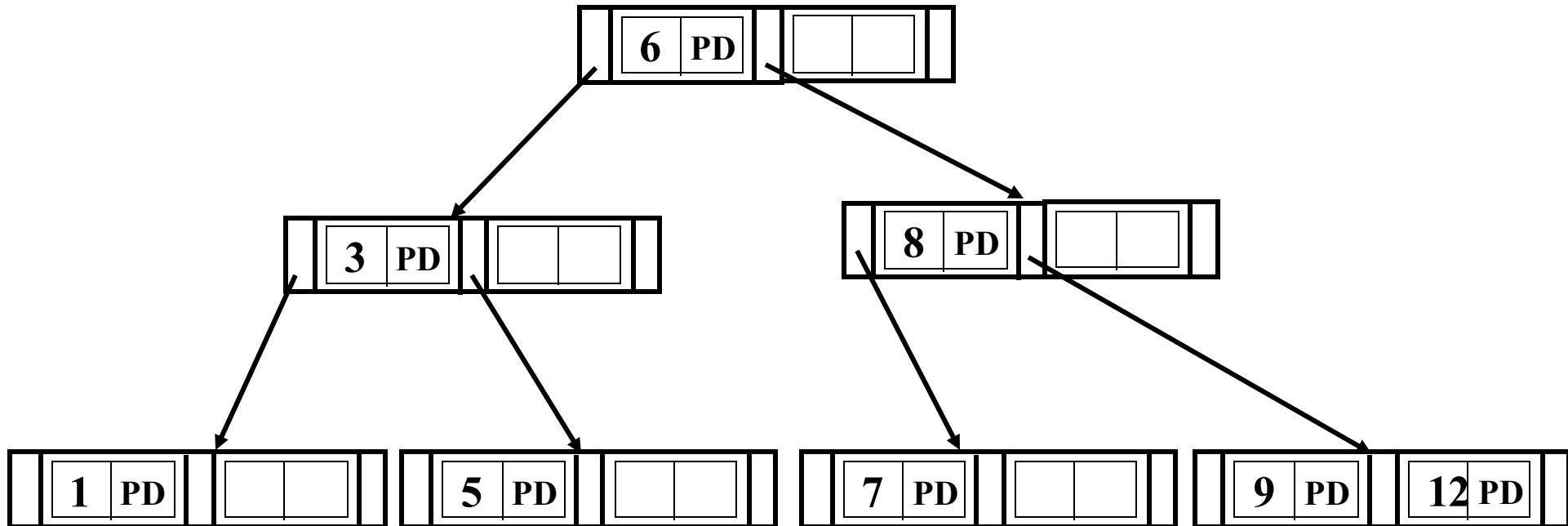
□ 1,3,5,6,7,8,9,12



Indexação

- Estruturas de Índices Ordenados -

- Árvores **B** (cont.)
 - Construa árvore B de ordem 3 (cont.)



- Pior caso: nós com capacidade mínima.
- Deve ser considerado para cálculo da altura

□ Árvores **B** (cont.)

□ Altura

- Quanto maior o número de ponteiros de árvore por nó
 - Menor a altura da árvore
 - Árvore larga
- Altura deve ser calculada no pior caso para árvores B
 - Cada nó apresenta sua capacidade mínima
- Existe uma relação entre altura **h** e número de nós **M** (na altura h) em uma árvore B de ordem n

$$h=0; M=1$$

$$h=1; M=n/2 \quad (\text{quantidade mínima de nós apontados em } h)$$

$$h=2; M=(n/2)^2$$

$$h=3; M=(n/2)^3$$

⋮

$$h=k; M=(n/2)^k \Rightarrow \log_{n/2} M = \log_{n/2} (n/2)^k$$

$$\mathbf{h = \log_{n/2} M}$$

- Quanto maior a ordem, menor a altura da árvore B

- Estrutura de Índices Ordenados (cont.)
 - **Árvores B^+ (B^+ -trees)**
 - Árvore balanceada
 - Altura da árvore é constante
 - Regras de formação de uma B^+ -Tree de ordem n
 - Cada nó interno de uma B^+ -tree é da forma
$$\langle PT_1, K_1, PT_2, K_2, \dots, PT_{m-1}, K_{m-1}, PT_m \rangle$$
 - $m \leq n$
 - Em um nó não folha,
 - Cada PT_i representa um ponteiro para uma subárvore (B^+ -tree)
 - Em um nó folha
 - Cada PT_i representa um ponteiro para a página que contém uma tupla r cuja chave de busca é igual a K_i
 - PT_m representa um ponteiro para a próxima folha

Indexação

- Estruturas de Índices Ordenados -

- Estrutura de Índices Ordenados (cont.)
 - Árvores **B⁺** (cont)
 - Regras de formação de uma *B⁺-Tree* de ordem ***n*** (cont.)
 - Dentro de um nó: $K_1 < K_2 < \dots < K_{m-1}$
 - Para todo valor *X* de chave de busca na subárvore apontada por **PT_i**, nós temos
$$K_{i-1} < X \leq K_i \text{ para } 1 < i < m, X \leq K_i \text{ para } i=1 \text{ e } K_{i-1} < X \text{ para } i=m$$
 - Cada nó não folha
 - Possui no máximo ***n*** ponteiros de árvore
 - Um nó com ***m*** ponteiros de árvore possui ***m-1*** valores de chave de busca
 - Possui no mínimo $\lceil n/2 \rceil$ ponteiros de árvore
 - $\lfloor (n-1)/2 \rfloor$ valores de chave de busca
 - Todos os nós exceto a raiz
 - A raiz possui no mínimo dois ponteiros de árvore

- Estrutura de Índices Ordenados (cont.)
 - Árvores **B⁺** (cont.)
 - Regras de formação de uma *B⁺-Tree* de ordem ***n*** (cont.)
 - Cada nó folha
 - Possui no máximo ***n*-1** ponteiros de dados e **um** ponteiro para próxima folha
 - Uma folha com ***m*** ponteiros de árvore possui ***m*-1** valores de chave de busca
 - Possui no mínimo $\lceil n/2 \rceil$ ponteiros de dados
 - $\lfloor (n-1)/2 \rfloor$ valores de chave de busca
 - Todos os nós exceto a raiz
 - A raiz possui no mínimo dois ponteiros de árvore
 - Todos nós folhas apresentam mesma altura

Indexação

- Estruturas de Índices Ordenados -

- Estrutura de Índices Ordenados (cont.)
 - **Árvores B^+** (cont.)
 - Uma entrada <chave de busca, ponteiro para página> só é inserida nos nós folhas
 - Cada nó não folha contém apenas valores de chave de busca e ponteiros de árvore
 - Não contém ponteiros de dados
 - Podem conter mais ponteiros de árvore que árvores B
 - A ordem de árvores B^+ é maior que a de árvores B
 - Para um mesmo tamanho de página T
 - Um nó em uma árvore B^+ pode ter mais filhos que um nó em uma árvore B
 - Qual uma importante conclusão que se pode tirar do fato acima??

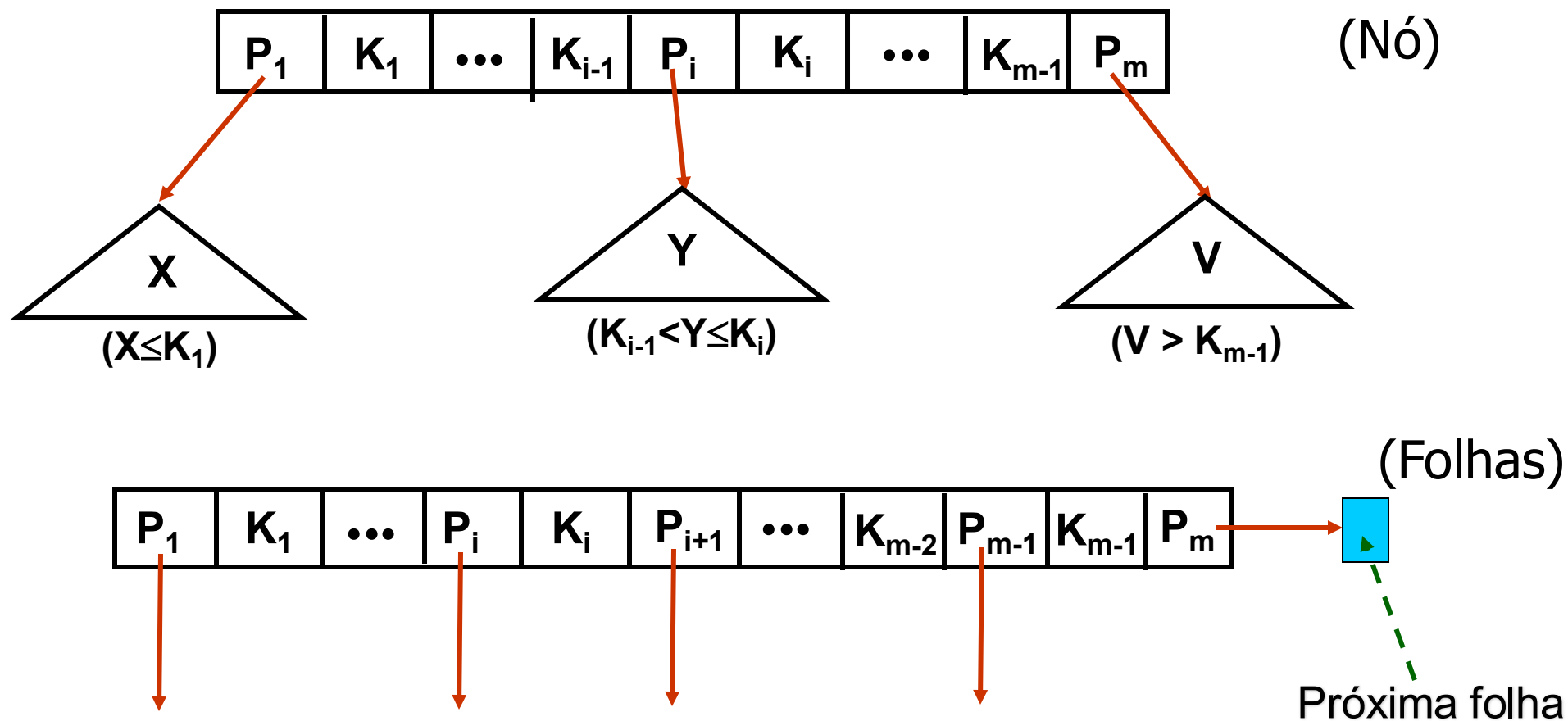
Indexação

- Estruturas de Índices Ordenados -

□ Estrutura de Índices Ordenados (cont.)

□ Árvores B^+ (cont.)

□ Estrutura de uma árvore B^+



- Estrutura de Índices Ordenados (cont.)

- Árvores **B⁺** (cont.)

- Algoritmo de inserção

- semelhante ao algoritmo para árvores B

- Quando há overflow ou underflow

- Só são transferidos para os nós não folhas valores de chave de busca

- Não são transferidos ponteiros de dados

- Cálculo da ordem n de uma árvore B⁺

- Parâmetros

- Tamanho de página □ **T** bytes

- Tamanho da chave de busca □ **C** bytes

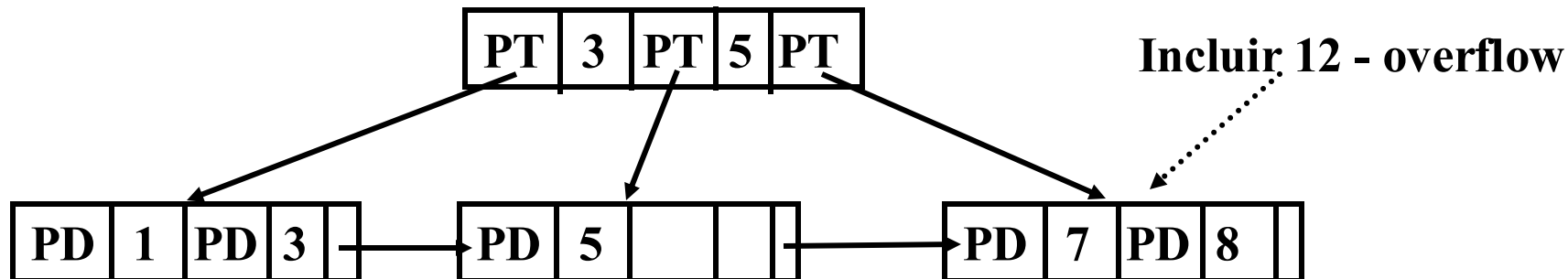
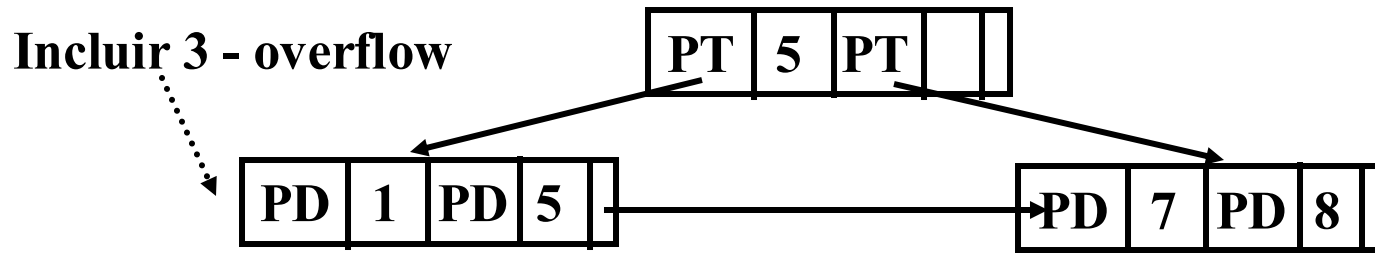
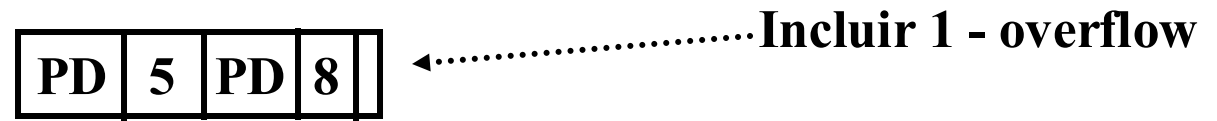
- Tamanho do ponteiro de árvore □ **P** bytes

- $(n * P) + ((n - 1) * C) \leq T$

Indexação

- Estruturas de Índices Ordenados -

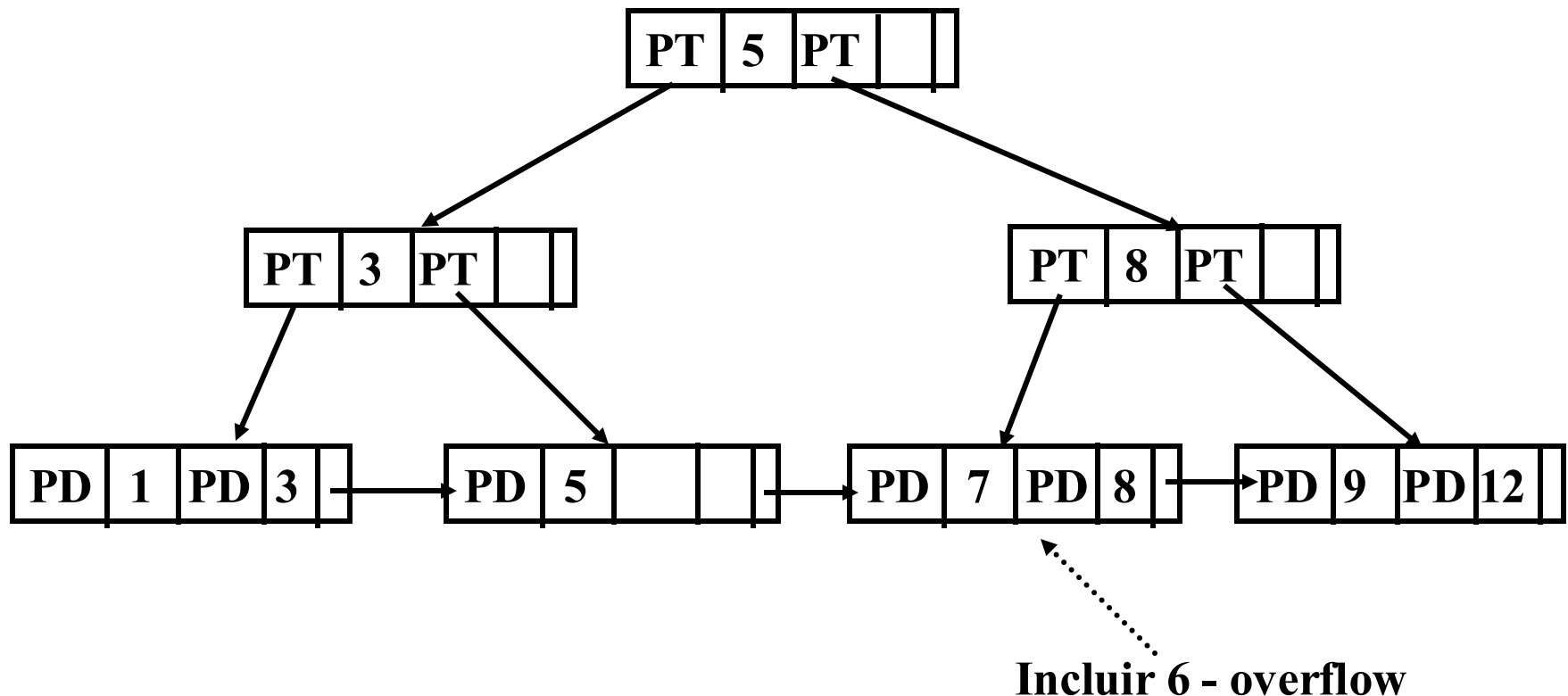
- Estrutura de Índices Ordenados (cont.)
 - Construa uma árvore **B⁺** de ordem 3 com a seguinte ordem de inserção: 8,5,1,7,3,12,9,6



Indexação

- Estruturas de Índices Ordenados -

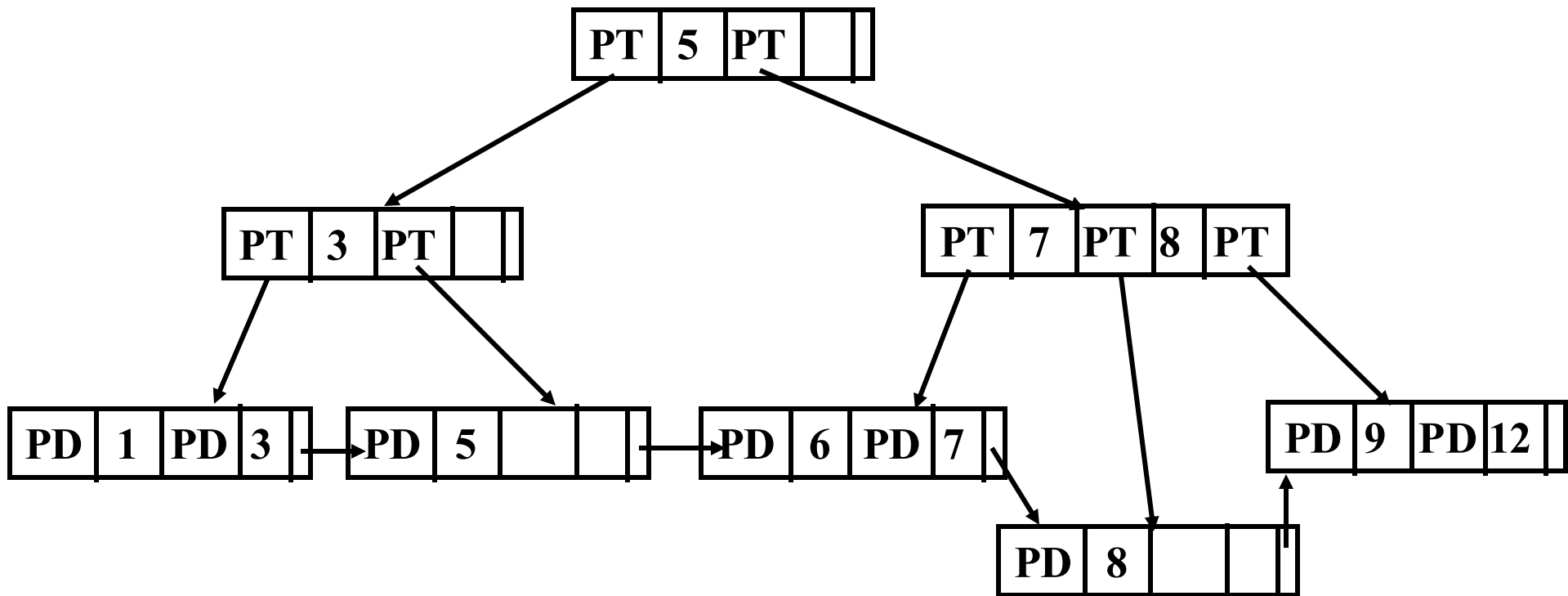
- Estrutura de Índices Ordenados (cont.)
 - Construa uma árvore **B⁺** de ordem 3 com a seguinte ordem de inserção: 8,5,1,7,3,12,9,6



Indexação

- Estruturas de Índices Ordenados -

- Estrutura de Índices Ordenados (cont.)
 - Construa uma árvore **B⁺** de ordem 3 com a seguinte ordem de inserção: 8,5,1,7,3,12,9,6



Indexação

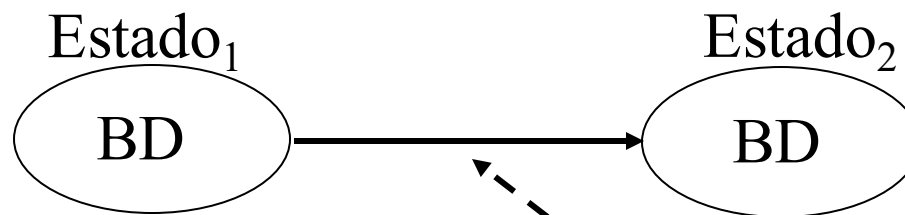
- Criação de Índices em SQL -

- ❑ Os comandos para criação de índices possuem sintaxes levemente diferentes para cada SGBD
- ❑ Sintaxe genérica
 - ❑ CREATE INDEX nome_indice ON tabela(colunas)
- ❑ Exemplos
 - ❑ Criação de índice primário no SQL Server
 - ❑ CREATE **CLUSTERED** INDEX in_depto_codigo ON Departamentos(codigo)
 - ❑ Criação de índice hash no PostgreSQL
 - ❑ CREATE INDEX in_func_matricula ON Funcionarios **USING hash** (matricula)
- ❑ Estude a documentação do seu SGBD!

Processamento de Transações

- O Problema de Concorrência em BDs -

- Banco de Dados
 - Estado
 - Transição de estado
 - Restrições de consistência (integridade)
 - Estado consistente de um banco de dados



Programas com operações sobre objetos de DB

Processamento de Transações


- O Problema de Concorrência em BDs -

- Concorrência em um ambiente multiusuário
 - Entrelaçamento (*interleaving*) de operações
 - Operações de um programa podem ser executadas entre duas operações de outro programa
 - Alterações inconsistentes no banco de dados
 - Transições de estado gerando estados inconsistentes do BD
 - Transições incorretas

Processamento de Transações

- O Problema de Concorrência em BDs -

❑ Transições Incorretas

Programa-1	Programa-2	
select saldo into : <i>valor</i> from tab_conta where num_conta=12		 Tempo
<i>valor=valor+500</i>	select saldo into : <i>valor</i> from tab_conta where num_conta=12	
update tab_conta set saldo= : <i>valor</i> where num_conta=12	<i>valor=valor-100</i>	
	update tab_conta set saldo= : <i>valor</i> where num_conta=12	

Saldo inicial (conta 12)= 400

Saldo produzido no BD=300 ⚡

Saldo correto final= 800

Lost update

Processamento de Transações

- O Problema de Concorrência em BDs -

- ❑ Entrelaçamento de operações de programas distintos
 - ❑ Pode provocar inconsistências no BD
- ❑ SGBD precisa monitorar e controlar a execução concorrente de programas
- ❑ **Controle de Concorrência** (Scheduler)

Processamento de Transações

- O Paradigma de Transação em BDs -

□ Transação

- Abstração que representa a sequência de operações de bancos de dados resultantes da execução de um programa
- sequência de *reads* e *writes*

begin transaction

declare @valor real

*select @valor=saldo from tab_conta
where num_conta=12*

set @valor=@valor+500

*update tab_conta set saldo= @valor
where num_conta=12*

commit

- $T_1 = r_1(\text{conta_12})w_1(\text{conta_12})$

Processamento de Transações

- O Paradigma de Transação em BDs -

□ Propriedades de uma Transação

□ Modelo **ACID**

□ Atomicidade

- A transação deve ser considerada indivisível
- Ou tudo é feito, ou nada!

□ Consistência

- Uma transação deve levar o BD de um estado consistente a outro estado consistente
- Todas as restrições de integridade devem ser respeitadas

Processamento de Transações

- O Paradigma de Transação em BDs -

□ Propriedades de uma Transação

□ Modelo **ACID**

□ Isolamento

- A execução de uma transação não deve ser incomodada pela execução de outra transação
- A transação deve ter a ilusão de que está sendo executada isoladamente

□ Durabilidade

- Os efeitos de uma transação commitada devem permanecer no BD
 - Ainda que aconteçam falhas!

- Cada SGBD possui seus próprios comandos para controle de transações
- Usualmente
 - BEGIN|START TRANSACTION
 - Inicia a transação
 - Transação ficará ativa no SGBD até que um comando de término seja executado
 - COMMIT
 - Termina a transação com sucesso
 - Todas as atualizações efetuadas pela transação devem persistir “para sempre” (durabilidade)

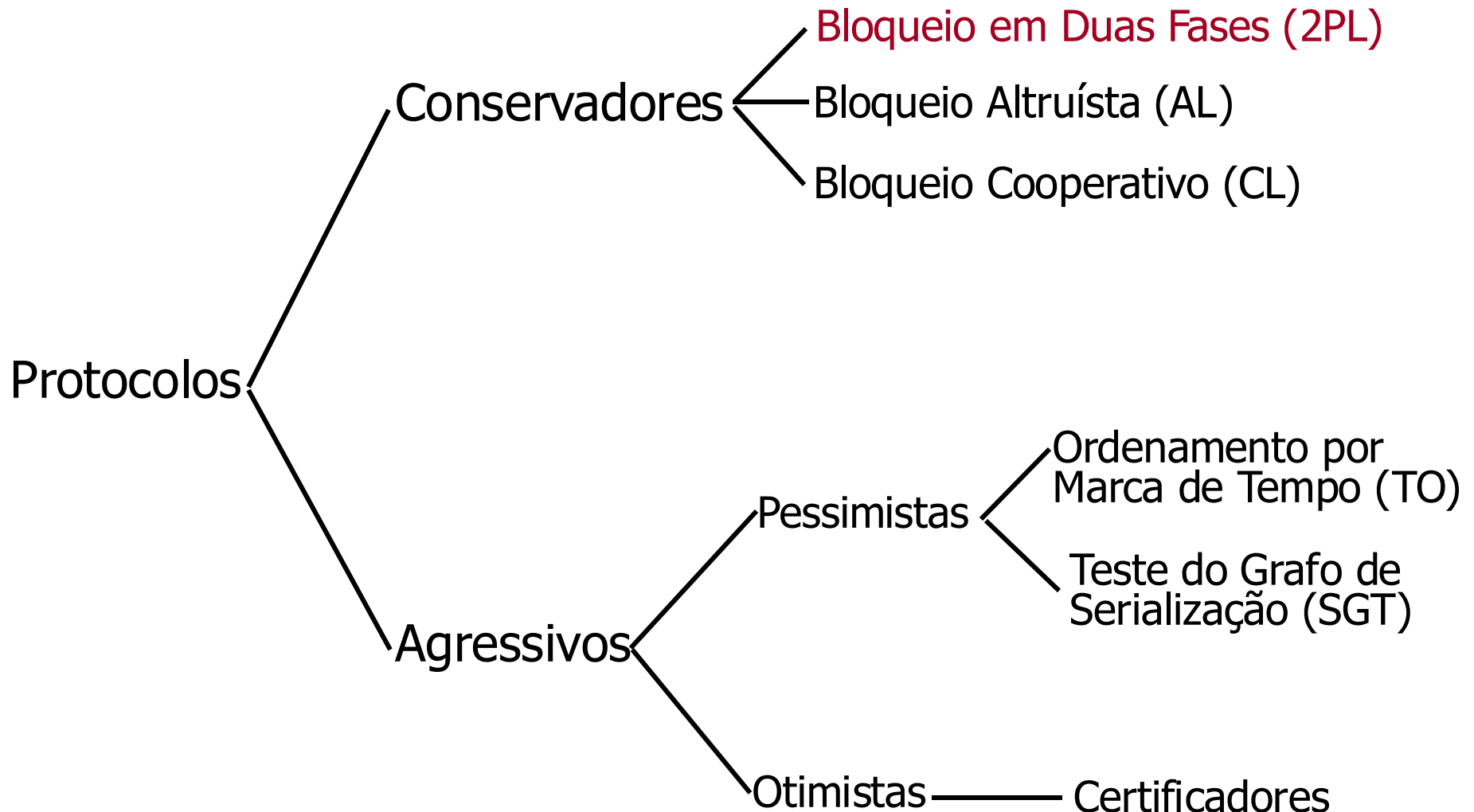
- Usualmente

- ROLLBACK

- Termina a transação com falha

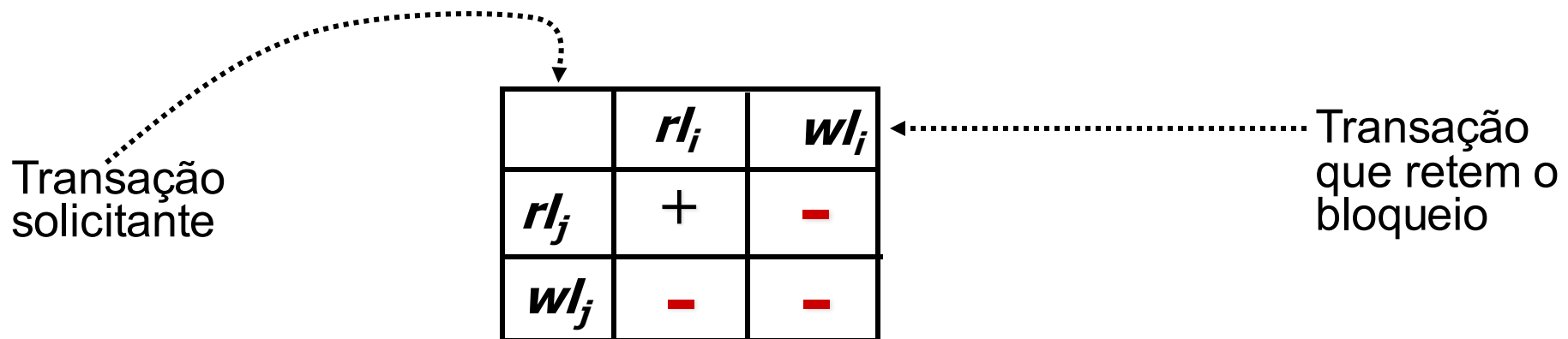
- Todas as atualizações efetuadas pela transação devem desfeitas, como se nunca tivessem existido (atomicidade)

- Modelos de processamento de transações são implementados através de protocolos para o controle de concorrência
- Responsáveis pela produção de schedules com base em um
 - critério de corretude e um grau de confiabilidade
- Entrada: operações pertencentes a transações distintas
- Saída: Seqüência corretamente sincronizada
- Núcleo de um scheduler



- Conservadores
 - Baseados em mecanismos de bloqueio
 - Atrasam a execução de operações para sincroniza-las corretamente
- Agressivos
 - Sincronizam operações imediatamente
 - Pessimistas
 - Decidem se aceitam ou rejeitam a operação
 - Se rejeitam, abortam a transação
 - Otimista
 - Aceitam a operação imediatamente
 - Periodicamente verificam a corretude do schedule já produzido

- Bloqueio em Duas Fases - **2PL** (*two-phase lock*)
 - Um tipo de bloqueio associado a cada objeto do BD
 - Granularidade de bloqueio: objeto
 - Tipo de bloqueio é definido pela operação a ser executada
 - Bloqueio de leitura (*read lock* - **rl**)
 - Bloqueio de escrita (*write lock* - **wl**)
 - Liberação de bloqueio
 - Compatibilidade de Bloqueios



? Protocolo

1. Seja $p_i \in \{r_i, w_i\}$. Para conceder um bloqueio do tipo $pl_i(x)$:
O scheduler verifica se existe um bloqueio incompatível com $pl_i(x)$.
Se existir, então
o scheduler **atrasa** a execução da operação $p_i(x)$,
até que $pl_i(x)$ seja concedido.
2. Uma vez uma transação T_i libere algum bloqueio, T_i não pode obter mais bloqueios sobre qualquer objeto do BD

? Duas fases

Primeira Fase: Aquisição de bloqueios

Segunda Fase: Liberação de bloqueios

+ Implementado pelos SGBDs existentes

□ Interface de Transação

- Gera o TRID para cada nova transação

□ Gerenciador de Bloqueios

- Executa as operações de bloqueio e liberação
- Gerencia duas estruturas de dados

□ Arquivo de Bloqueios

- Contém informação sobre os bloqueios

□ Cada entrada

- **TRID**
- **Tipo-Bloqueio**
- **OID**

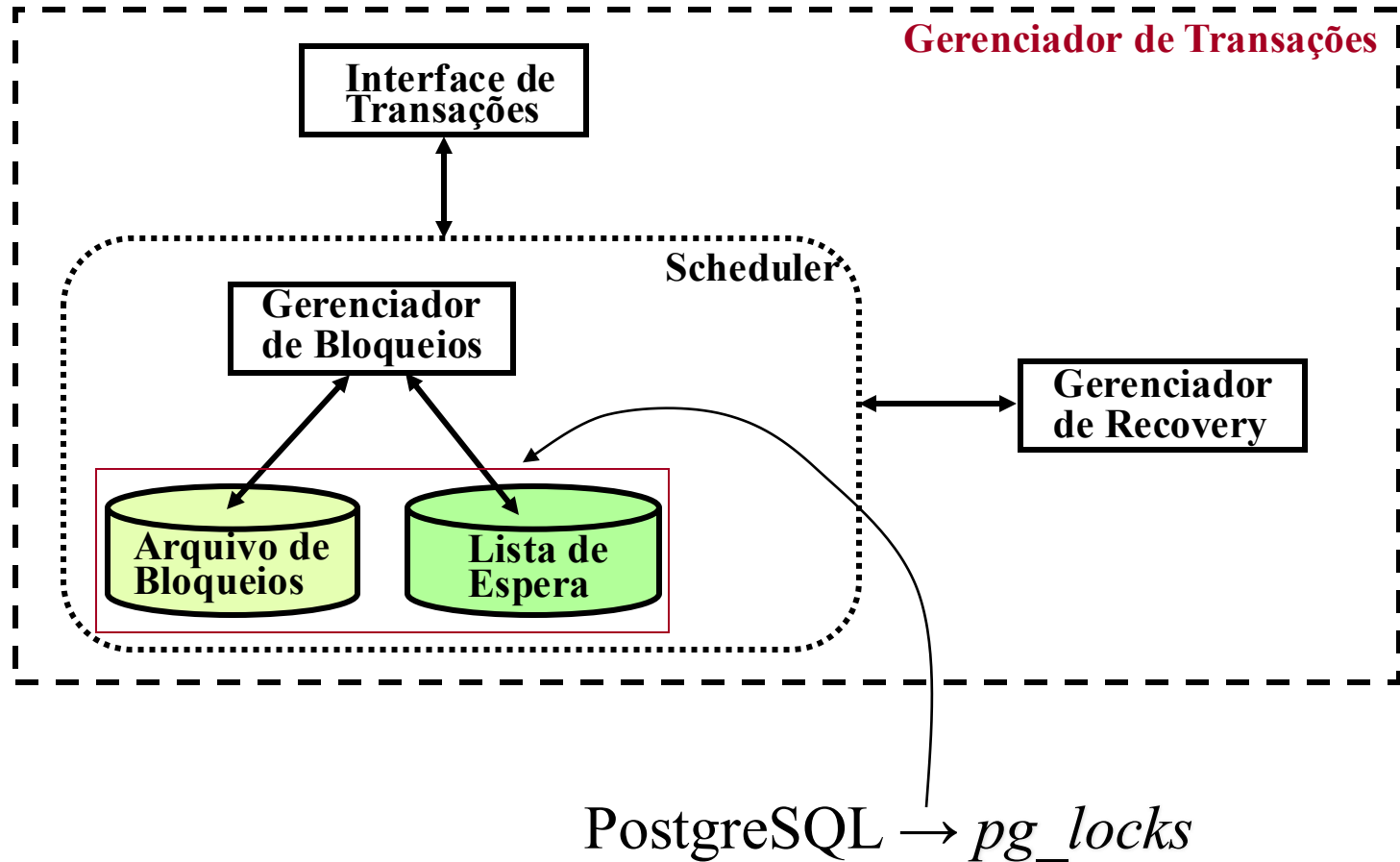
□ Lista de Espera

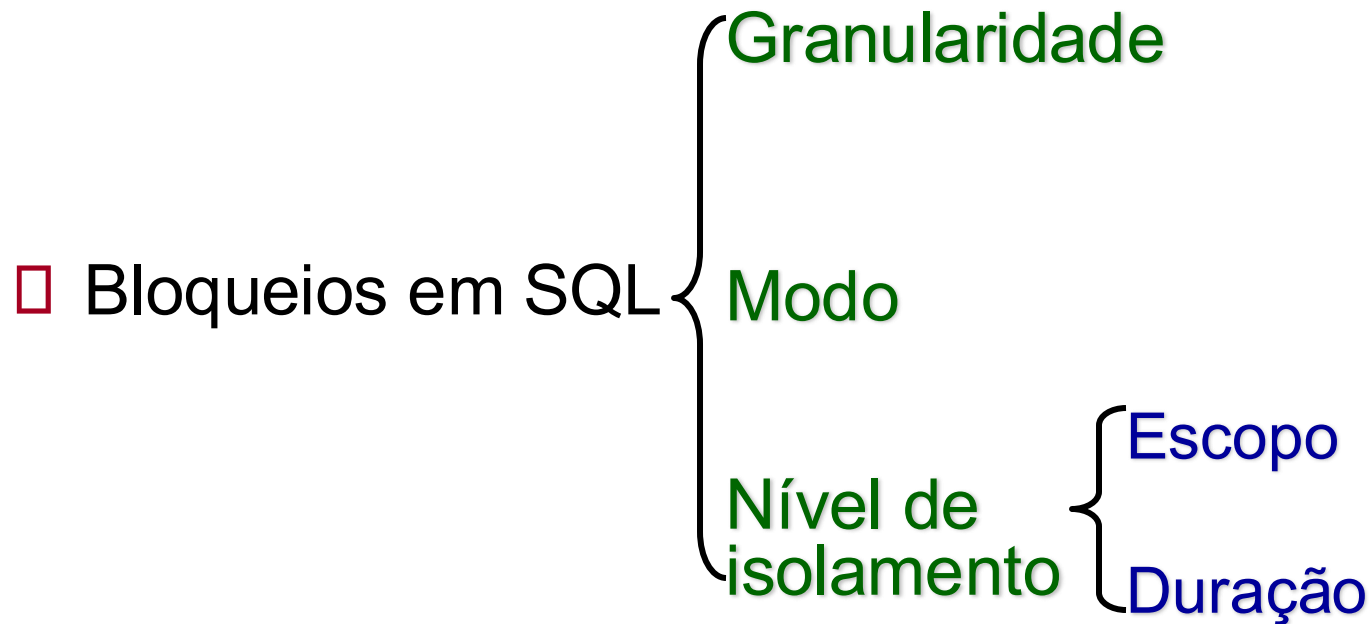
- Contem dados das transações bloqueadas

□ Gerenciador de Recovery (recuperação)

Protocolos para o Controle de Concorrência

- Arquitetura de GTs que utilizam 2PL -





□ Granularidade de bloqueios

□ DB2

□ Tablespace

□ Table

□ Page

□ Row

□ Definição

create tablespace ...

locksize row

page

table

tablespace

any

Maior grau de concorrência

Menor overhead para gerenciamento de bloqueios

Definição dinâmica pelo DBMS

- Granularidade de Bloqueios (Cont.)

- SQL Server

- Banco de dados

- Tabela

- Extensão (*extent*)

- Conjunto de oito páginas contíguas

- Chave de busca (*key*)

- Bloqueio de uma tupla pela estrutura de índice

- Utilizado para bloquear tuplas por intervalo de chaves de busca

- Tupla (*RID*)

- Modo de bloqueio
 - **Leitura** (share - S)
 - **Escrita** (exclusive - X)
- Nível de isolamento define:
 - Escopo de bloqueio
 - **Objeto**
 - Bloqueio sobre uma tupla específica da tabela
 - Acessadas via Cursor
 - **Predicado**
 - Bloqueio sobre tuplas que satisfazem um predicado
 - Tuplas existentes no banco de dados
 - Tuplas ainda não existentes
 - + Satisfariam o predicado caso fossem incluídas no banco de dados
 - Tuplas a serem alteradas e que passariam a satisfazer o predicado após a alteração

- Nível de isolamento define:
 - Duração do bloqueio
 - Longa
 - O bloqueio é retido até após o *commit* ou *abort*
 - Exemplo
 - ... $rl_i(x \text{ in } P)r_i(x \text{ in } P) \dots c_i$
 - Nenhuma outra transação T_j pode inserir ou atualizar objetos que satisfaçam predicado P , até que T_i libere o bloqueio de leitura sobre P
 - + Final da transação
 - Curta
 - O bloqueio é liberado imediatamente após a execução da operação associada ao bloqueio
 - Exemplo
 - ... $rl_i(x)r_i(x)\text{release_}rl_i(x) \dots c_i$
 - ... `Select * from Empregado where salario>499 ... Commit`

□ Nível de isolamento define:

□ Duração do bloqueio

□ **Curta**

□ Exemplo – Cursor

BOT ... Select * from Empregado where salario>499,... Commit

Estrutura de Cursor



05	André	5000	1
08	Bárbara	2000	2
10	Inês	7000	1
15	Sofia	500	1
16	Natalya	2000	2
20	Caio	500	1
22	Lucas	500	1
30	Rebeca	550	2

1) A tupla (05,'André', 5000, 1) não está bloqueada, pois o bloqueio sobre objetos (tuplas) é de curta duração

- Níveis de Isolamento

- **read uncommitted**

- Bloqueios de leitura

- Não necessário

- Bloqueios de escrita

- Curta duração tanto para objetos (tuplas de uma tabela) como predicados

- **read committed**

Default: Oracle e SQL Server

- Bloqueios de leitura

- Curta duração tanto para objetos (tuplas) como predicados

- Bloqueios de escrita

- Longa duração tanto para objetos como predicados

□ Níveis de Isolamento (cont.)

□ read committed

□ Anomalias

□ **Nonrepeatable read**

$$S_1 = r_2(x)w_1(x, 2)c_1r_2(x)c_2$$

Transação T_2 está lendo valores diferentes de x

+ Deve ser evitado !

- Improvável de acontecer (duas leituras de T_2 sobre x)

□ **Lost update**

$$S_2 = r_2(x)r_1(x)w_1(x, x+20)c_1w_2(x, x+10)r_2(z)c_2$$

Operação de escrita $w_1(x)$ foi sobreposta por $w_2(x)$

+ Deve ser evitado !

- Suponha que T_1 e T_2 executavam operações de depósitos bancários

□ Níveis de Isolamento (cont.)

□ cursor stability

□ Bloqueios de leitura

- Curta duração para objetos (tuplas)

- Curta duração para predicados

- Bloqueio no item corrente do cursor é mantido até

- O cursor seja movido

- O cursor seja fechado

□ Bloqueios de escrita

- Longa duração tanto para objetos como predicados

□ Evita a anomalia *lost update*

$$S_3 = r_2(x)w_1(x)c_1w_2(x)r_2(z)c_2$$

schedule não produzido
por cursor stability
(produzido por read committed)

bloqueio liberado após
a execução de $r_2(z)$, quando
o cursor será movimentado

- Níveis de Isolamento (cont.)
 - cursor stability
 - DB2
 - SQL Server
 - Na clausula DECLARE CURSOR
 - Opção SCROLL_LOCKS

□ Níveis de Isolamento (cont.)

□ repeatable read

□ Bloqueios de leitura

□ Longa duração para objetos (tuplas)

□ Curta duração para predicados

□ Bloqueios de escrita

□ Longa duração tanto para objetos como predicados

□ Exemplo

□ BOT ... Select * from Empregado where salario>499 ... C

Estrutura de cursor



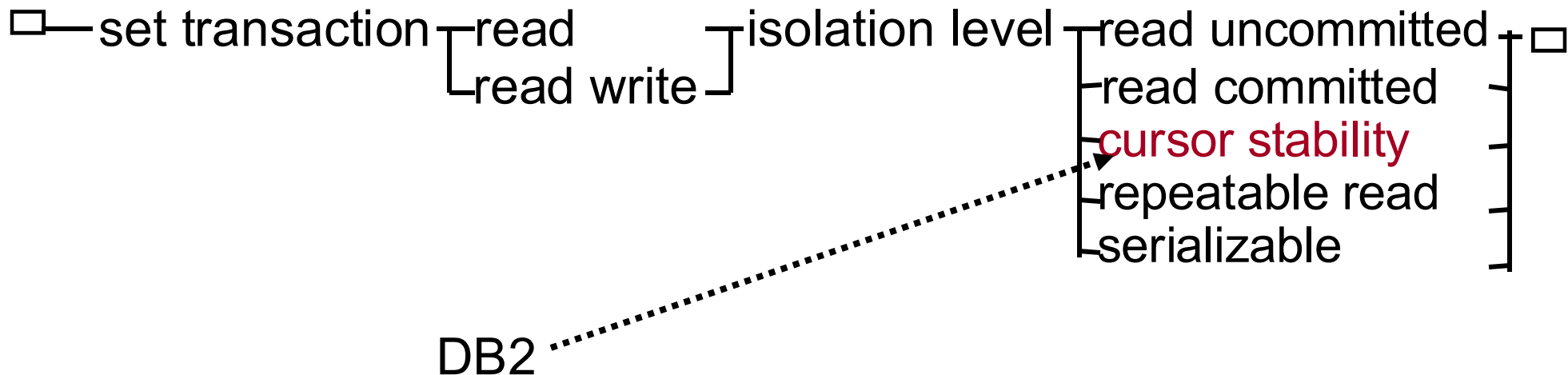
05	André	5000	1
08	Bárbara	2000	2
10	Inês	7000	1
15	Sofia	500	1
16	Natalya	2000	2
20	Caio	500	1

1) Na tabela, pode ser inserida uma nova tupla (07,'José',4000,1)

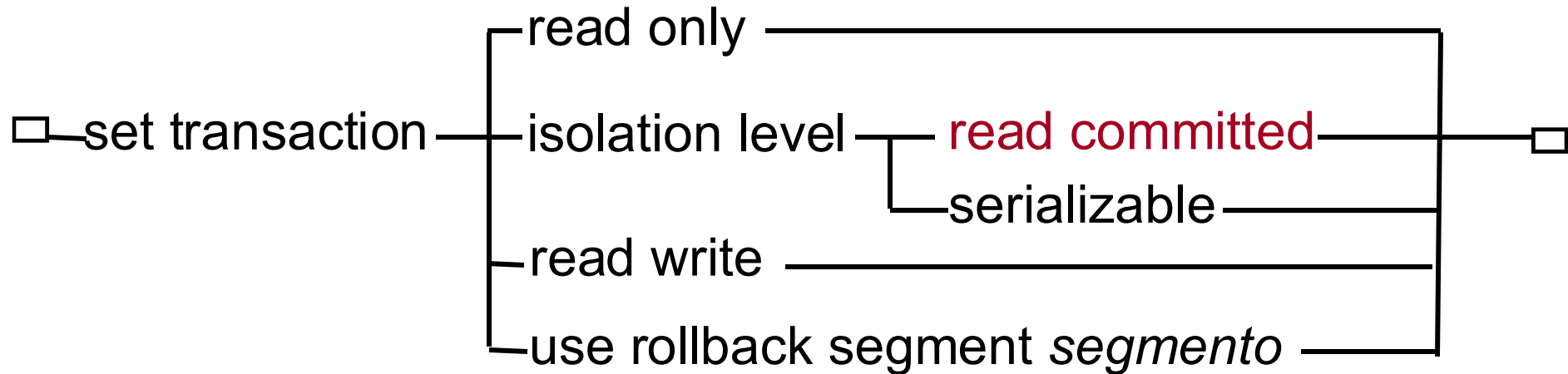
2) A tupla (05,'André', 5000, 1) ficará bloqueada até a execução do commit da transação.

- Níveis de Isolamento (cont.)
 - **serializable (serializability)**
 - Bloqueios de leitura
 - Longa duração tanto para objetos como predicados
 - Bloqueios de escrita
 - Longa duração tanto para objetos como predicados

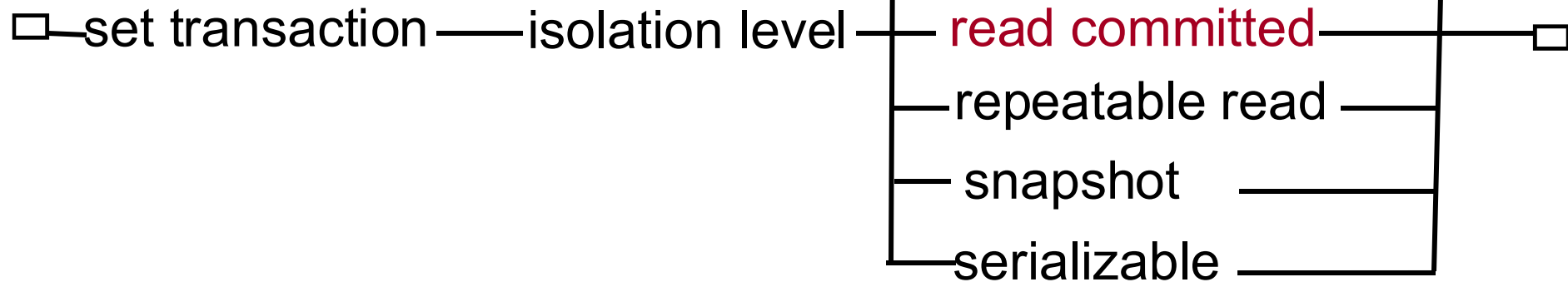
□ Sintaxe SQL99



□ Sintaxe Oracle



□ Sintaxe SQL Server 2005



□ Snapshot

□ Implementa o protocolo 2V2PL

□ Ligar opção ALLOW_SNAPSHOT_ISOLATION

- Escolha do nível de isolamento
 - Importante decisão de projeto
 - Decidir entre
 - Garantir consistência dos objetos do banco de dados
 - Anomalias devem ser evitadas
 - Nonrepeatable read, Lost update, etc ...
 - Garantir um alto grau de concorrência

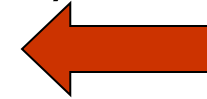
Processamento de Consultas

- Conceitos Básicos -

- Consulta (*query*) em BDs
 - Expressão de linguagem que descreve dados a serem recuperados do banco de dados
- Linguagem de Consulta
 - Linguagem de alto nível
 - Além de consultar:
 - Definir a estrutura de dados (DDL)
 - Especificar restrições de integridade
 - Modificar dados no BD (DML)
 - Especificar restrições de segurança (DCL)
 - Prover controle de concorrência (DTL)

- Linguagens comerciais (SGBDs Relacionais)

- SQL (*Structured Query Language*)



SELECT d.nome

FROM Departamento d, Empregado e

WHERE d.cod_dep=e.cod_dep

- Quel (*Query Language*)

RANGE OF (d,e) IS (Departamento, Empregado)

RETRIEVE INTO nome(d.nome)

WHERE (d.cod_dep=e.cod_dep)

- QBE (Query By Example)

- Processamento de Consulta

- Conjunto de atividades envolvidas na extração de dados do BD

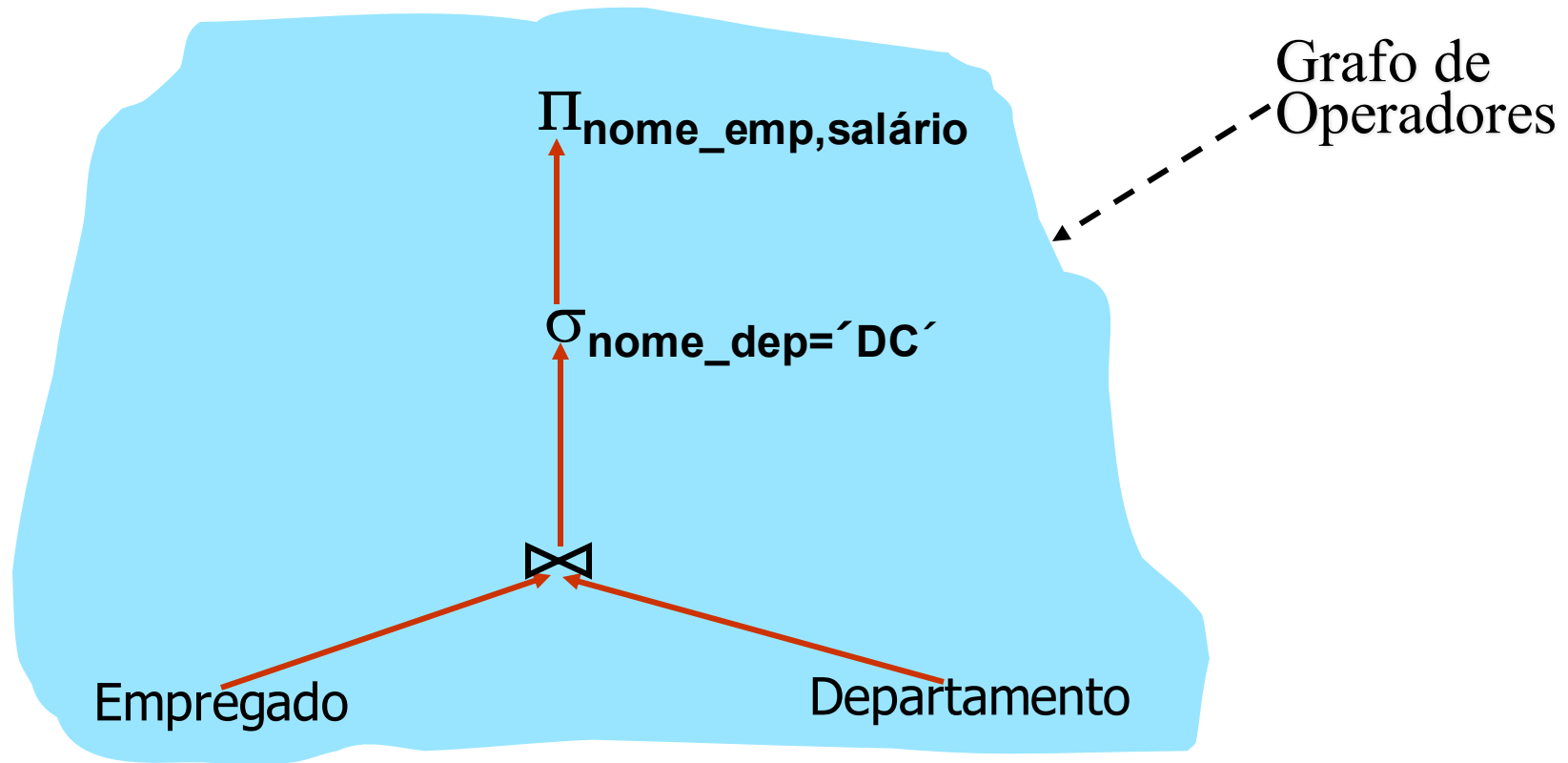
- Consultas em linguagem de alto nível
 - Mapeadas para outras formas de representação
 - Suporte para garantir transformações de consultas (otimização)
 - Expressar uma grande classe de consultas
- Forma adotada
 - A consulta em SQL é convertida para uma expressão da Álgebra Relacional
 - A expressão da Álgebra Relacional é representada como um Grafo de Operadores

□ Grafo de Operadores

- Descreve um fluxo de dados com base na execução dos operadores da Álgebra Relacional
- Nós representam os operadores da consulta
 - Representam as tabelas sobre as quais a consulta é executada
- Arestas indicam a direção do fluxo de dados
- Representa uma estratégia de execução para a construção da relação resultado
- Implementado nos SGBDs existentes

Consulta:

Nome e salário dos empregados que trabalham no departamento DC



- Construa o grafo de operadores para as seguintes consultas:
 - Nome, salário e endereço do departamento de lotação dos empregados que trabalham no departamento DC e ganham mais que 8000
 - ```
select e.nome, d.nome
from Departamento d inner join
 (select nome, lotação from Empregado where
 salário>700) e on e.lotação=d.cod_depart
```

- Informação necessária para cálculo de custos de consultas relacionais
  - $n_r$  representa cardinalidade da relação  $r$
  - $p_r$  representa o número de páginas contendo as tuplas de  $r$
  - $s_r$  indica o tamanho (em bytes) das tuplas de  $r$
  - $f_r$  indica o número de tuplas de  $r$  por página (fator de página)
  - $V(A, r)$  representa o número de valores distintos de  $A$  em  $r$   
*Se  $A$  é uma chave, qual o valor de  $V(A, r)$ ?*
  - $FS(A, r)$  indica a média do número de tuplas que satisfazem uma condição de igualdade sobre o atributo  $A$  (fator de seletividade)  
*Se  $A$  é uma chave, qual o valor de  $FS(A, r)$ ?*
  - $HT_i$  indica o número de níveis da estrutura de índice  $i$
- Medida de custo
  - **Número de páginas transferidas**



### □ *File Scan*

- Localizar e recuperar tuplas que satisfazem a condição de seleção

- Tuplas armazenadas em um mesmo arquivo

### □ Busca linear

- Tabela é lida seqüencialmente

- Todas tuplas são testadas

- Não pressupõe *sort* ou existência de índices

- Estimativa de Custo  **$EC = p_r$**

- $p_r$  tamanho da tabela em páginas

- *File Scan* (cont.)

- Busca binária

- Tabela ordenada pelo atributo da condição (**igualdade**)

- Busca binária para primeira página

- Depois busca linear

- $EC = \log_2(p_r) + \lceil FS(A,r) / f_r \rceil - 1$

*Primeira página contendo tuplas que satisfazem a condição*

*Páginas ocupadas por  $FS(A,r)$*

*Número de tuplas da tabela que satisfazem a condição*

### □ Índice Primário (cont.)

#### □ Estimativa de custo (Igualdade)

- Em atributos do tipo *chave* (definidos como índice primário)

$$EC = HT_i + 1$$

- Em atributos não *chave* (definidos como índice primário)

$$EC = HT_i + \lceil FS(A,r) / f_r \rceil$$

Tuplas em páginas contíguas

### □ Índice Secundário

#### □ Índice que não é primário

#### □ Estimativa de custo (Igualdade)

- Em atributos do tipo *chave* (definidos como índ. secundário)

$$EC = HT_i + 1$$

- Em atributos não *chave*

$$EC = HT_i + FS(A,r)$$

Pior caso, tuplas em páginas diferentes

### □ Predicado de seleção determina o acesso através de índices



- Índice Hash
  - EC=2
    - Considerando a não ocorrência de *bucket overflow*

### □ *Nested-Loop Join* ( $r \bowtie s$ )

```
para cada tupla t_r em r faça
 para cada tupla t_s em s faça
 se o par (t_r, t_s) satisfaz a condição de join
 então
 inclua $t_r t_s$ ao resultado
 Fim;
Fim;
```

$$\square EC = p_r + (n_r * p_s)$$

### □ Exercício

□ Departamento e Empregado apresentam cardinalidade igual a 50 e 1000, respectivamente. Considere ainda que  $f_D=10$  e  $f_E=5$

□ Calcule o custo de  $\text{Empregado} \bowtie \text{Departamento}$

$$\square p_D = 50/10 \text{ e } p_E = 1000/5; EC = 5200$$



- *Nested-Loop Join* utilizando índices
  - Definição de índice para o atributo de junção na tabela mais interna (**s**)
  - $EC = p_r + (n_r * c)$ , onde  
*c* representa o custo de uma seleção simples
- Exercício
  - As tabelas Departamento e Empregado apresentam cardinalidades igual a 50 e 1000 respectivamente. Considere que  $f_E = 5$  e que foi definido um índice primário para o atributo de Cod\_dep em Departamento.
  - Suponha que a estrutura de índice foi implementada através de uma árvore B<sup>+</sup> com altura igual 3
  - Calcule o custo de Empregado ⋈ Departamento
  - $p_E = 1000/5$  e  $c = 3 + 1$       $EC = 4200$

### □ *Block Nested-Loop Join* ( $r \bowtie s$ )

```
para cada bloco B_r da tabela r faça
 para cada bloco B_s da tabela s faça
 para cada tupla t_r em B_r faça
 para cada tupla t_s em B_s faça
 se o par (t_r, t_s) satisfaz a condição de join
 então
 inclua $t_r t_s$ ao resultado
 Fim;
 Fim;
 Fim;
Fim;
```

□  $EC = p_r + (p_r * p_s)$

□ **Pior caso:** carregar apenas uma página de cada tabela por vez no buffer

□ Independentes de ordenação das tabelas



### □ Exercício

- No algoritmo *block nested-loop join*, a ordem das tabelas pode influir na estimativa de custo (considere as tabelas Departamento e Empregado) ?
  - **A tabela menor deve se posicionar no loop mais externo**
- Qual será a estimativa de custo para o algoritmo *block nested-loop join*, caso pelo menos uma das tabelas poder ser carregada por inteiro no buffer (memória) ?
  - $EC = p_r + p_s$



### □ Merge-Join ( $r \bowtie s$ )

#### □ Pré-requisitos:

- Critério de junção com igualdade (equijoin)
- **r** e **s** devem estar ordenadas pelos atributos de junção
- Uma das colunas do critério de junção deve ser chave

#### □ Algoritmo:

*ler primeira tupla de **s***

*para cada tupla **t<sub>r</sub>** em **r** faça*

*enquanto (**t<sub>r</sub>**[atr\_join] ≥ **t<sub>s</sub>**[atr\_join]) e ¬fim(**s**) faça*

*se o par (**t<sub>r</sub>**, **t<sub>s</sub>**) satisfaz o critério de junção*

*então*

*inclua **t<sub>r</sub>t<sub>s</sub>** no resultado;*

*ler próxima tupla **t<sub>s</sub>** em **s***

*fim;*

*fim;*

#### □ $EC = p_r + p_s$

### □ Sort-Merge-Join ( $r \bowtie s$ )

□ **r** e **s** não precisam estar ordenadas pelo atributo de junção

□ Algoritmo:

se **r** não estiver ordenada por seu atributo de junção então ordene **r**

se **s** não estiver ordenada por seu atributo de junção então ordene **s**

ler primeira tupla de **s**

para cada tupla  $t_r$  em **r** faça

enquanto  $(t_r[\text{atr\_join}] \geq t_s[\text{atr\_join}])$  e  $\neg \text{fim}(\mathbf{s})$  faça

se o par  $(t_r, t_s)$  satisfaz o critério de junção

então

inclua  $t_r t_s$  no resultado;

ler próxima tupla  $t_s$  em **s**

fim;

fim;

□  $EC = p_r + p_s + o(r) + o(s)$

$o(r)$  = custo da ordenação de **r**

$o(s)$  = custo da ordenação de **s**

- *Hash-Join* ( $r \bowtie s$ )
  - Implementação de equijoin
  - Ideia básica
    - Particionar tuplas das relações em *buckets*
      - Com base em valores de uma função de hash aplicada sobre os atributos de junção
  - Funcionamento
    - *Dada a função*
      - $h: \text{atr\_junção} \longrightarrow \{0, 1, \dots, \text{max}\}$
    - Comparar tuplas das tabelas  $r$  e  $s$  para as quais
      - $h(r.\text{atr\_junção}) = h(s.\text{atr\_junção})$

□ *Hash-Join* ( $r \bowtie s$ )

□  $h: \text{atr\_junção} \longrightarrow \{0, 1, \dots, \text{max}\}$

□  $H_{r_0}, H_{r_1}, \dots, H_{r_{\text{max}}}$  representam as partições das tuplas de  $r$  (inicialmente vazias)

□ Cada tupla  $t_r \in r$  é inserida na partição  $H_{r_i}$ , onde  
 **$i = h(t_r[\text{atr\_junção}])$**

□  $H_{s_0}, H_{s_1}, \dots, H_{s_{\text{max}}}$  representam as partições das tuplas de  $s$  (inicialmente vazias)

□ Cada tupla  $t_s \in s$  é inserida na partição  $H_{s_i}$ , onde  
 **$i = h(t_s[\text{atr\_junção}])$**

### □ Hash-Join ( $r \bowtie s$ )

- Criando partições para as tabelas

Para cada tupla  $t_r \in r$  faça

$i = h(t_r[\text{atr\_join}])$

$H_{r_i} = H_{r_i} \cup \{t_r\}$

Partições da tabela  $r$

.....

Para cada tupla  $t_s \in s$  faça

$i = h(t_s[\text{atr\_join}])$

$H_{s_i} = H_{s_i} \cup \{t_s\}$

Partições da tabela  $s$



### □ Hash-Join (cont.)

#### □ Calculando a junção

**de  $i=0$  até  $max$  faça**

ler  $H_{s_i}$  e construir **índice hash** para tuplas  $H_{s_i}$

**para cada tupla  $t_r$  em  $H_{r_i}$  faça**

utilize o índice de hash sobre  $H_{s_i}$  para localizar  
todas tuplas  $t_s$  onde  $t_s[atr\_join] = t_r[atr\_join]$

inclua  $t_r t_s$  ao resultado

**Fim;**

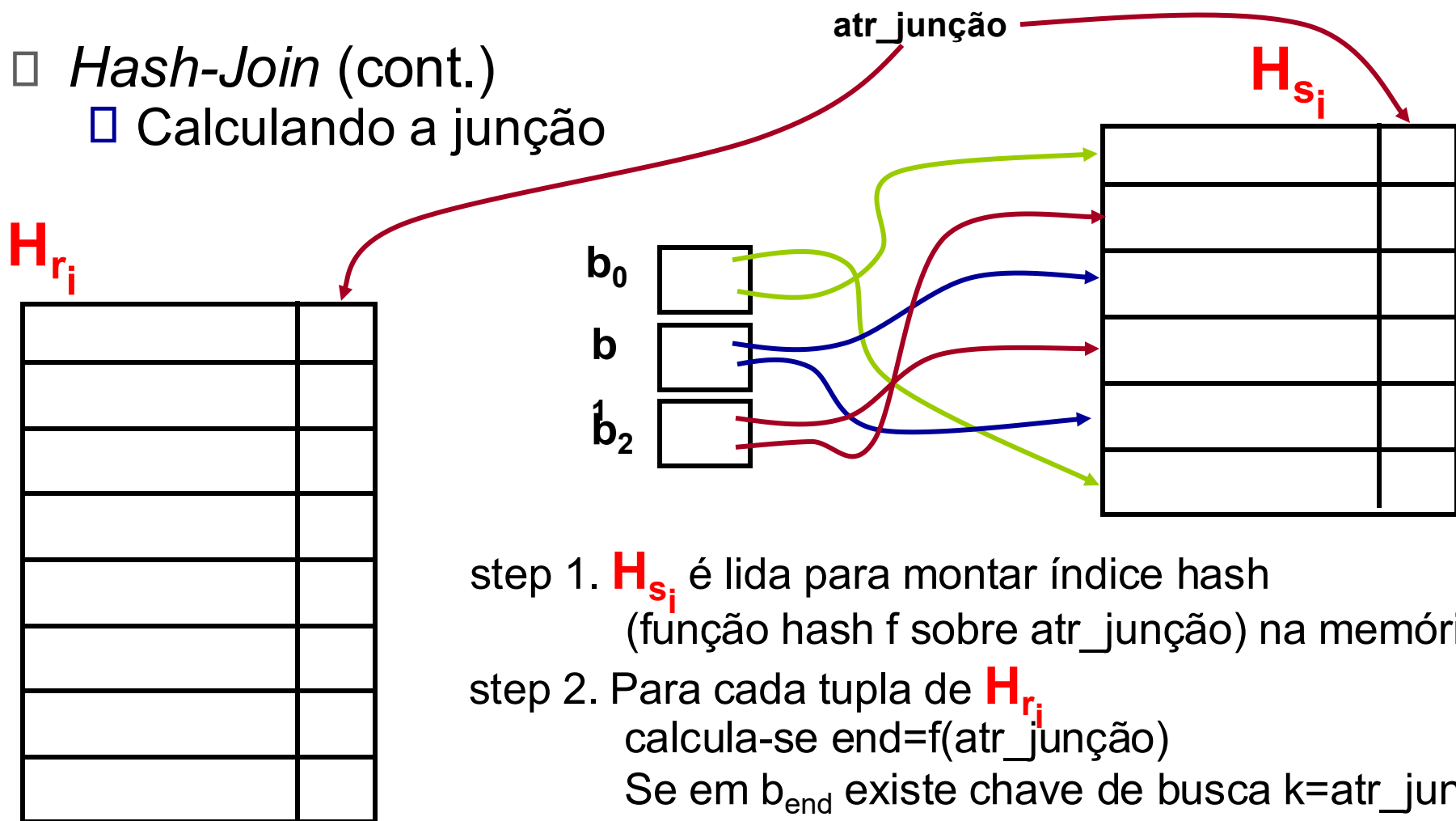
**Fim;**

Função hash sobre o  
atributo de junção, mas  
diferente de  $h$

nested-loop join utilizando índice (hash)  
entre  $H_{r_i}$  e  $H_{s_i}$

### □ Hash-Join (cont.)

#### □ Calculando a junção



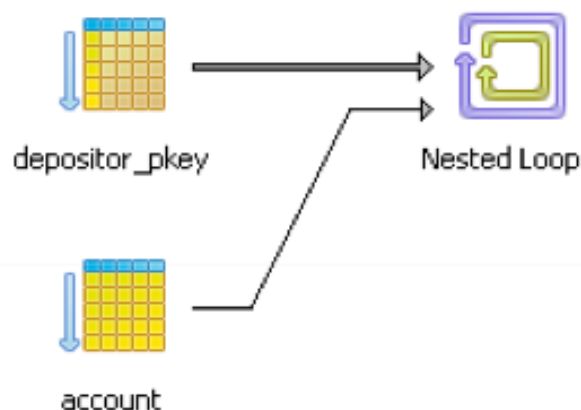
step 1.  $H_{s_i}$  é lida para montar índice hash  
(função hash  $f$  sobre  $atr\_junção$ ) na memória

step 2. Para cada tupla de  $H_{r_i}$   
calcula-se  $end = f(atr\_junção)$   
Se em  $b_{end}$  existe chave de busca  $k = atr\_junção$   
Então recuperar tupla com  $atr\_junção = k$

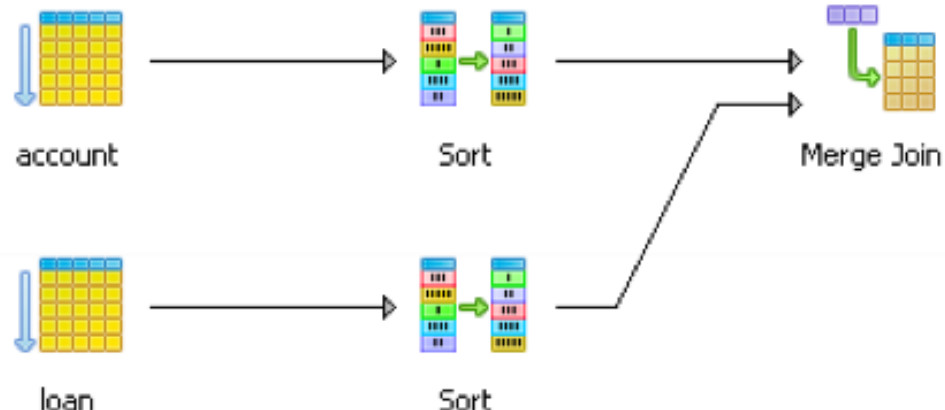


- Hash-Join (cont.)
  - Estimativa de custos
    - Custo para construir partições (ler+gravar)
      - $EC = 2*(pr + ps)$
    - Custo para calcular junção
      - $EC = pr + ps$
    - Custo total
      - $EC = 3*(pr + ps)$

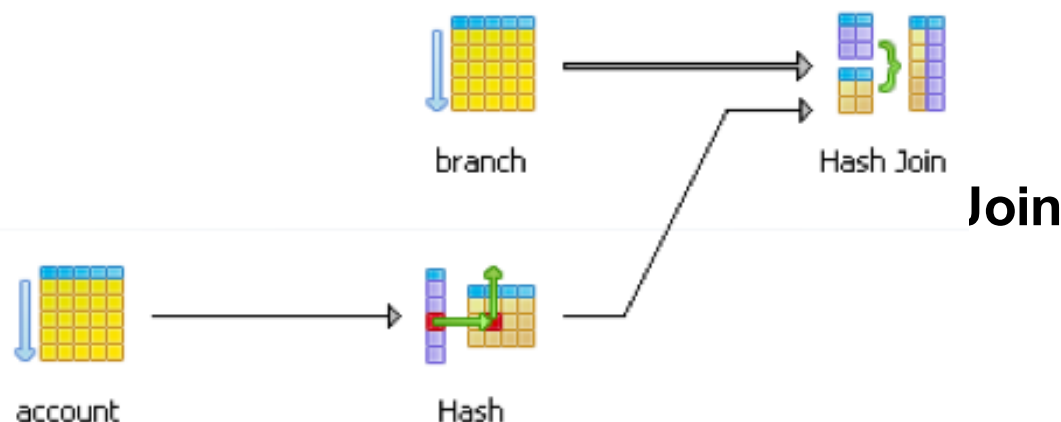
### Algoritmos de Junção em Planos do PostgreSQL



**Nested-Loop Join**



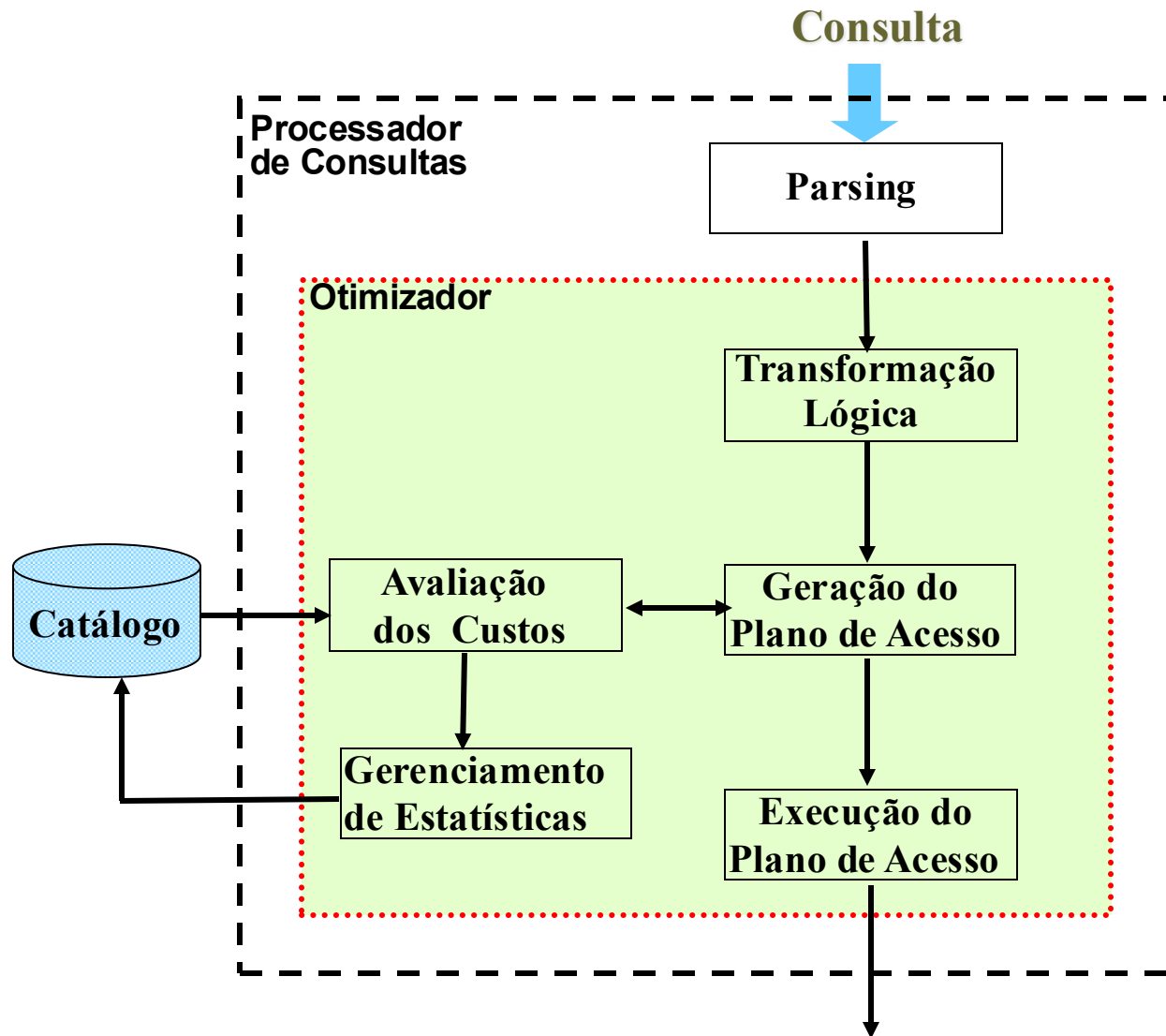
**Merge Join**



**Join**

# Processamento de Consultas

## - Fases do Processamento de Consultas -



- Parsing

- Analisador

- Sintático

- Semântico

- Mapeador

Consulta do usuário



Forma interna de representação  
(álgebra ou cálculo relacional)

- Otimização

- ❑ Objetivos
    - ❑ Minimizar o tempo de resposta
    - ❑ Minimizar a utilização de recursos (custos) para uma determinada saída
  - ❑ Custos a serem minimizados
    - ❑ Custo de Comunicação
      - ❑ Transmissão de dados
    - ❑ Custo de Acesso à Memória Secundária (disco)
      - ❑ Carregamento de páginas (disco) na memória principal
    - ❑ Custo de Armazenamento
      - ❑ Ocupação de *buffers*
    - ❑ Custo de Computacional
      - ❑ Utilização de CPU
- Peso maior em SGBDs centralizados*

- Transformação Lógica
  - Várias expressões equivalentes para uma mesma consulta em uma mesma forma de representação
  - Objetivos
    - Padronização
      - Construção de uma forma padronizada para o processo de otimização
    - Simplificação
      - Eliminação de redundância
    - Melhoria
      - Construção de expressões mais eficientes com relação a performance de execução



### ☐ Padronização

#### â Forma padronizada

##### ë Prenex normal form

$$\tilde{\text{a}} ((\exists/\forall \mathbf{t})(\mathbf{t} \in \mathbf{Rel} \wedge \mathbf{P}(\mathbf{t}))$$

+ Exemplo

$$\mathbf{Pred}_1 \wedge ((\exists \mathbf{t})(\mathbf{t} \in \mathbf{Rel} \wedge \mathbf{P}(\mathbf{t})) \Leftrightarrow ((\exists \mathbf{t})(\mathbf{t} \in \mathbf{Rel} \wedge (\mathbf{P}(\mathbf{t}) \wedge \mathbf{Pred}_1))$$

##### ë Disjunctive prenex normal form (DPNF)

$$\tilde{\text{a}} (P_{11} \wedge \dots \wedge P_{1n}) \vee (P_{21} \wedge \dots \wedge P_{2n}) \vee \dots \vee (P_{k1} \wedge \dots \wedge P_{kn})$$

##### ë Conjunctive prenex normal form (CPNF)

$$\tilde{\text{a}} (P_{11} \vee \dots \vee P_{1n}) \wedge (P_{21} \vee \dots \vee P_{2n}) \wedge \dots \wedge (P_{k1} \vee \dots \vee P_{kn})$$

### □ Simplificação

#### □ Eliminação de redundância

##### □ Aplicação das regras de idempotência

- $A \vee A \Leftrightarrow A$
- $A \wedge A \Leftrightarrow A$
- $A \vee \neg A \Leftrightarrow \text{True}$
- $A \wedge \neg A \Leftrightarrow \text{False}$
- $A \wedge (A \vee B) \Leftrightarrow A$
- $A \vee (A \wedge B) \Leftrightarrow A$
- $A \wedge \text{True} \Leftrightarrow A$
- $A \vee \text{False} \Leftrightarrow A$
- $A \vee \text{True} \Leftrightarrow \text{True}$
- $A \wedge \text{False} \Leftrightarrow \text{False}$

### □ Melhoria

- O processo de simplificação não produz expressões únicas
- Expressões não redundantes e semanticamente equivalentes podem ser diferentes quanto a parametros de performance
  - Expressões semanticamente equivalentes produzem o mesmo conjunto de tuplas (relação)
- Regras de transformação e heurísticas para encontrar a melhor expressão

### □ Melhoria (cont.)

#### □ Regras de transformação

$$\square \sigma_{\theta_1 \wedge \theta_2}(r) \Leftrightarrow \sigma_{\theta_1}(\sigma_{\theta_2}(r))$$

□ Qual a forma mais eficiente??

- Redução de resultados (tabelas) intermediários ou
- Deslocamento de seleção para baixo no grafo de consulta

$$\square \sigma_{\theta_1}(\sigma_{\theta_2}(r)) \Leftrightarrow \sigma_{\theta_2}(\sigma_{\theta_1}(r))$$

$$\square \Pi_{A_1}(\Pi_{A_2}(\dots (\Pi_{A_n}(r)) \dots)) \Leftrightarrow \Pi_{A_1}(r)$$

Cada  $A_i$  representa um conjunto de atributos

$$\square \sigma_{\theta_1}(r \bowtie_{\theta_2} s) \Leftrightarrow (\sigma_{\theta_1}(r)) \bowtie_{\theta_2} (s)$$

□ **Somente no caso em que a condição de seleção  $\theta_1$  consiste apenas de atributos de  $r$**

### □ Melhoria (cont.)

#### □ Regras de transformação

$$\square \sigma_{\theta_1 \wedge \theta_2}(r \bowtie_{\theta_3} s) \Leftrightarrow (\sigma_{\theta_1}(r)) \bowtie_{\theta_3} (\sigma_{\theta_2}(s))$$

□ **Somente no caso em que a condição de seleção  $\theta_1$  consiste apenas de atributos de  $r$  e a condição de seleção  $\theta_2$  consiste de atributos de  $s$**

$$\square (r \cup s) \bowtie_{\theta} (v \cup t) \Leftrightarrow (r \bowtie_{\theta} v) \cup (r \bowtie_{\theta} t) \cup (s \bowtie_{\theta} v) \cup (s \bowtie_{\theta} t)$$

### □ Melhoria (cont.)

#### □ Regras de transformação

$$\square \Pi_{A_1 \cup A_2}(r \bowtie_{\theta} s) \Leftrightarrow (\Pi_{A_1}(r)) \bowtie_{\theta} (\Pi_{A_2}(s))$$

□ **Somente quando  $A_1$  e  $A_2$  são atributos de  $r$  e  $s$  respectivamente e a condição de junção  $\theta$  consiste de atributos em  $A_1 \cup A_2$**

$$\square \Pi_{A_1 \cup A_2}(r \bowtie_{\theta} s) \Leftrightarrow \Pi_{A_1 \cup A_2}((\Pi_{A_1 \cup A_3}(r)) \bowtie_{\theta} (\Pi_{A_2 \cup A_4}(s)))$$

□  **$A_1$  e  $A_2$  são atributos de  $r$  e  $s$  respectivamente.  $A_3$  representa o conjunto de atributos de  $r$  envolvidos na condição de junção  $\theta$ , mas que não pertencem a  $A_1 \cup A_2$ . Por sua vez,  $A_4$  representa o conjunto de atributos de  $s$  envolvidos na condição de junção  $\theta$ , mas que não pertencem a  $A_1 \cup A_2$**

Cada  $A_i$  representa um conjunto de atributos

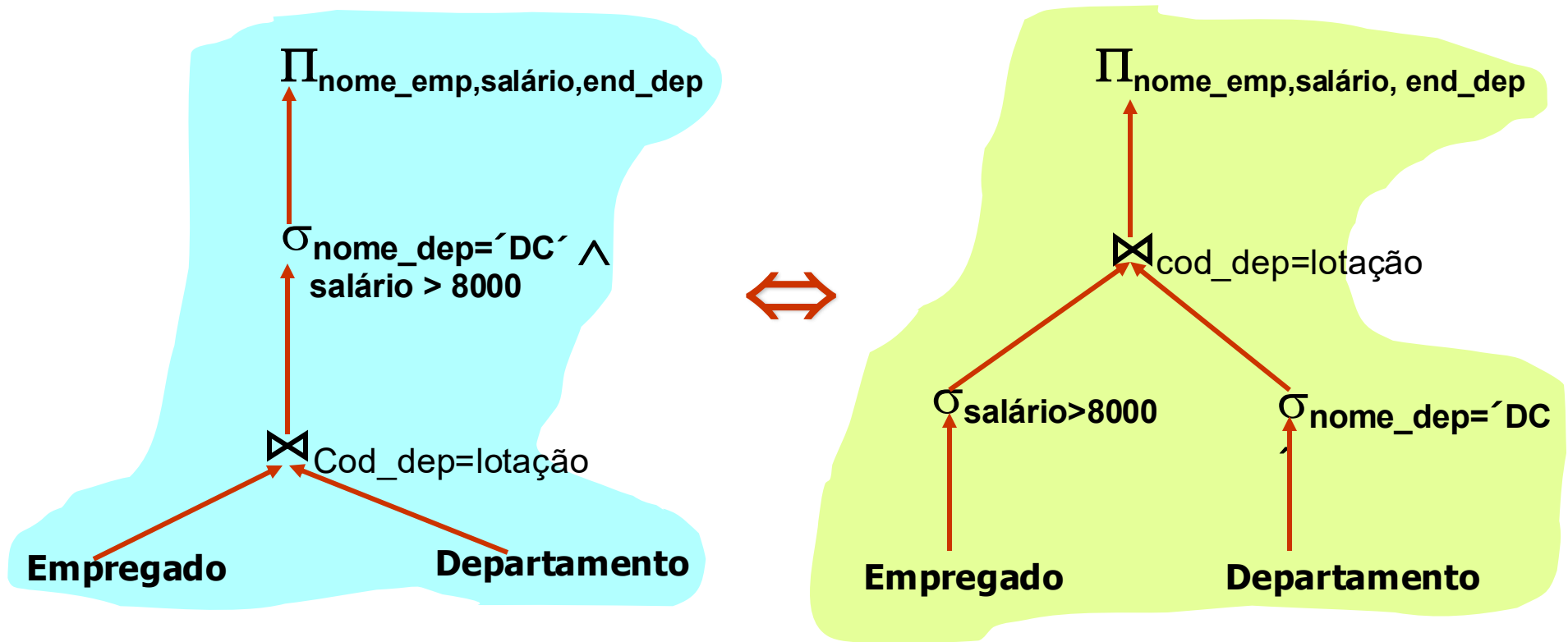
### □ Melhoria (cont.)

#### □ Heurística

- Mover operações seletivas (seleção e projeção) sobre operações construtivas (junção e produto cartesiano)
  - Executar operações seletivas antes que operações construtivas
- **Mover operações seletivas (seleção e projeção) para baixo no grafo de consulta**

Consulta:

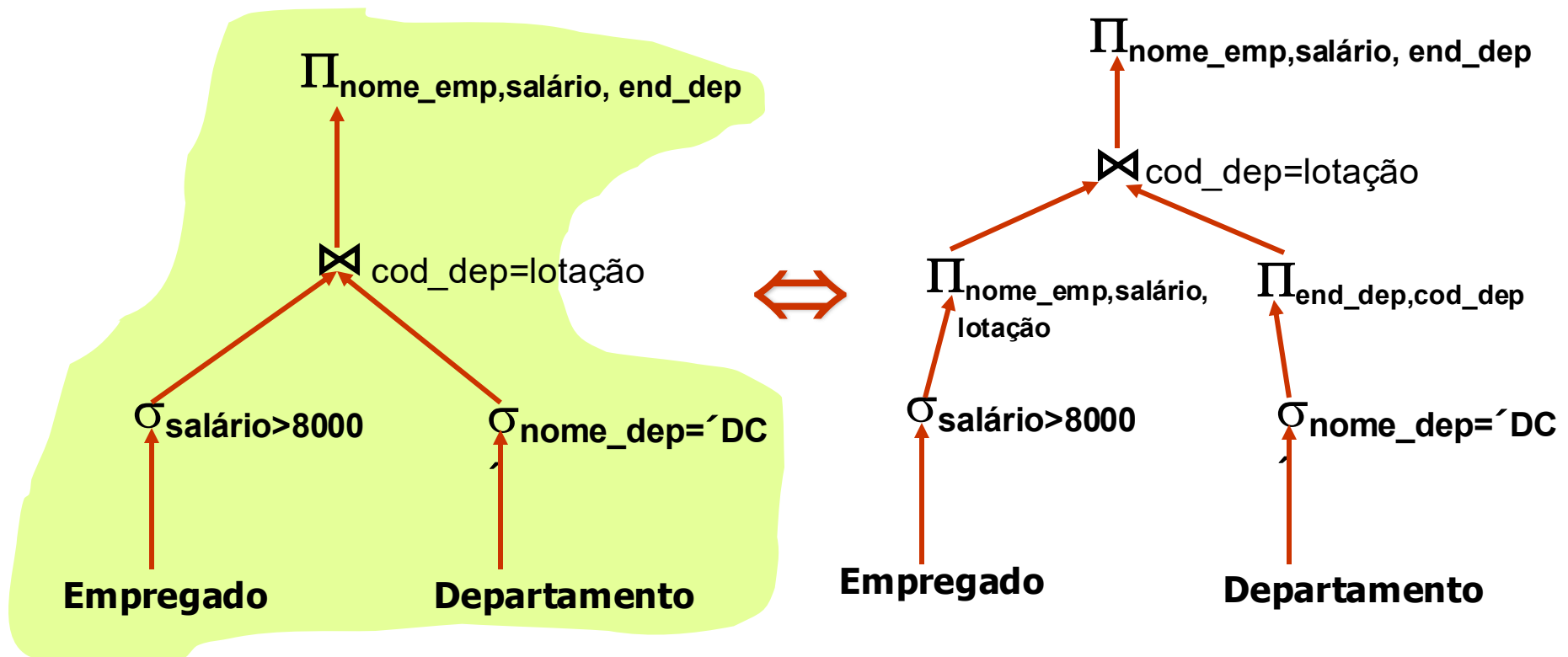
**Nome e salário dos empregados que trabalham no departamento DC e ganham mais que 8000, com o respectivo endereço do departamento**

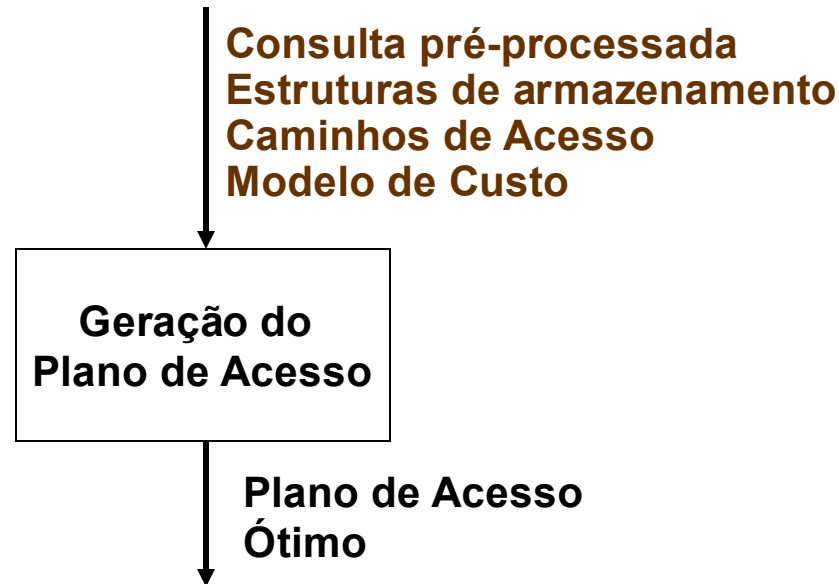




### Consulta (Cont.):

**Nome e salário dos empregados que trabalham no departamento DC e ganham mais que 8000, com o respectivo endereço do depart.**





### □ Plano de Acesso

- Descreve a seqüência de execução das operações de uma expressão (consulta), partindo de tabelas existentes até o resultado final

### □ Passos

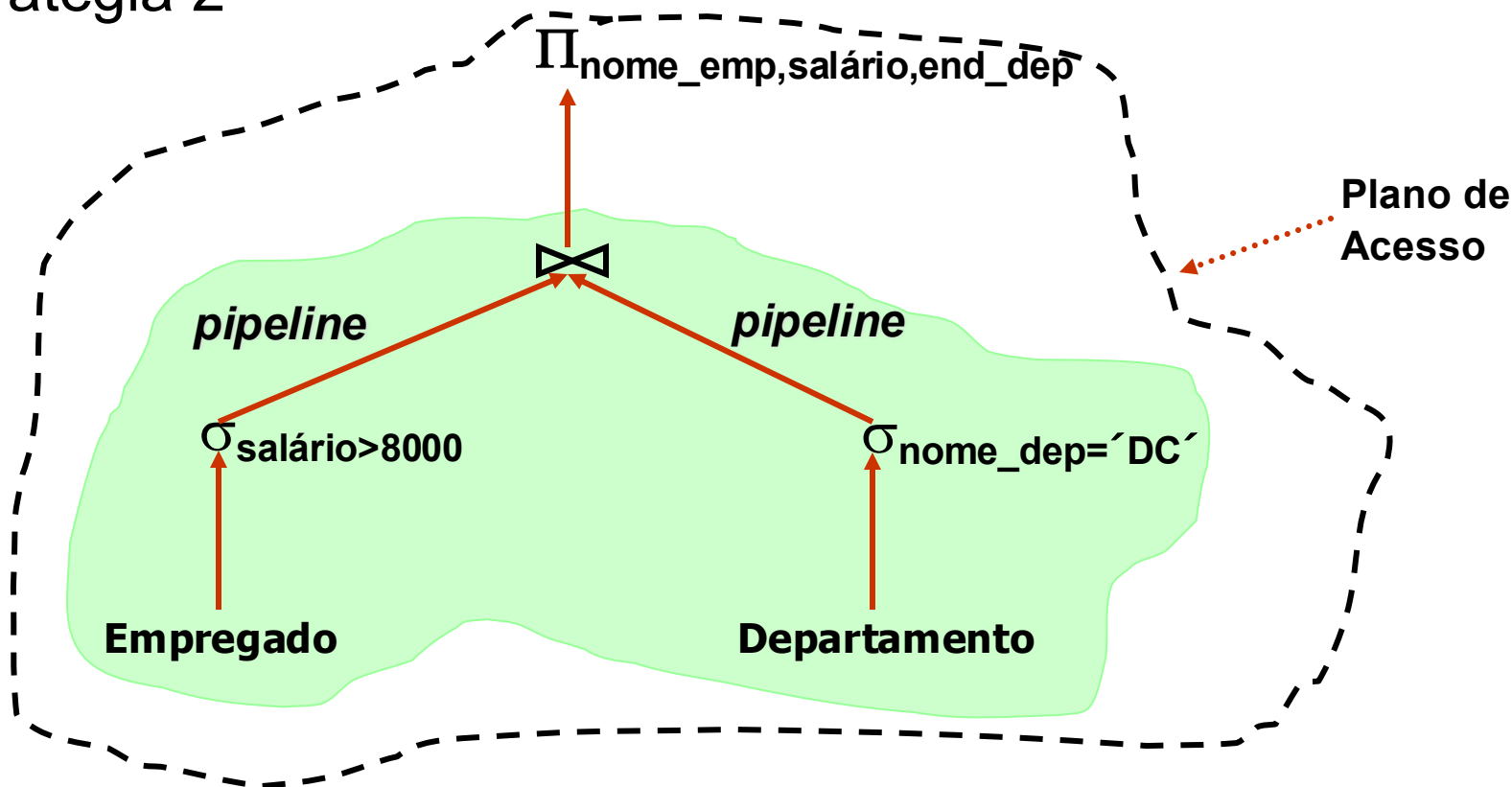
1. Gerar possíveis planos de acessos para a execução da consulta
2. Estender os planos de acesso com detalhes físicos dos dados (por exemplo, ordem de *sort*, existência de índices)
3. Escolher o plano mais barato com base no modelo de custo e custos de processamento

### □ O otimizador deve gerar um conjunto de planos de acesso

- grande o suficiente para conter o plano ótimo
- pequeno o suficiente para garantir custos aceitáveis de otimização

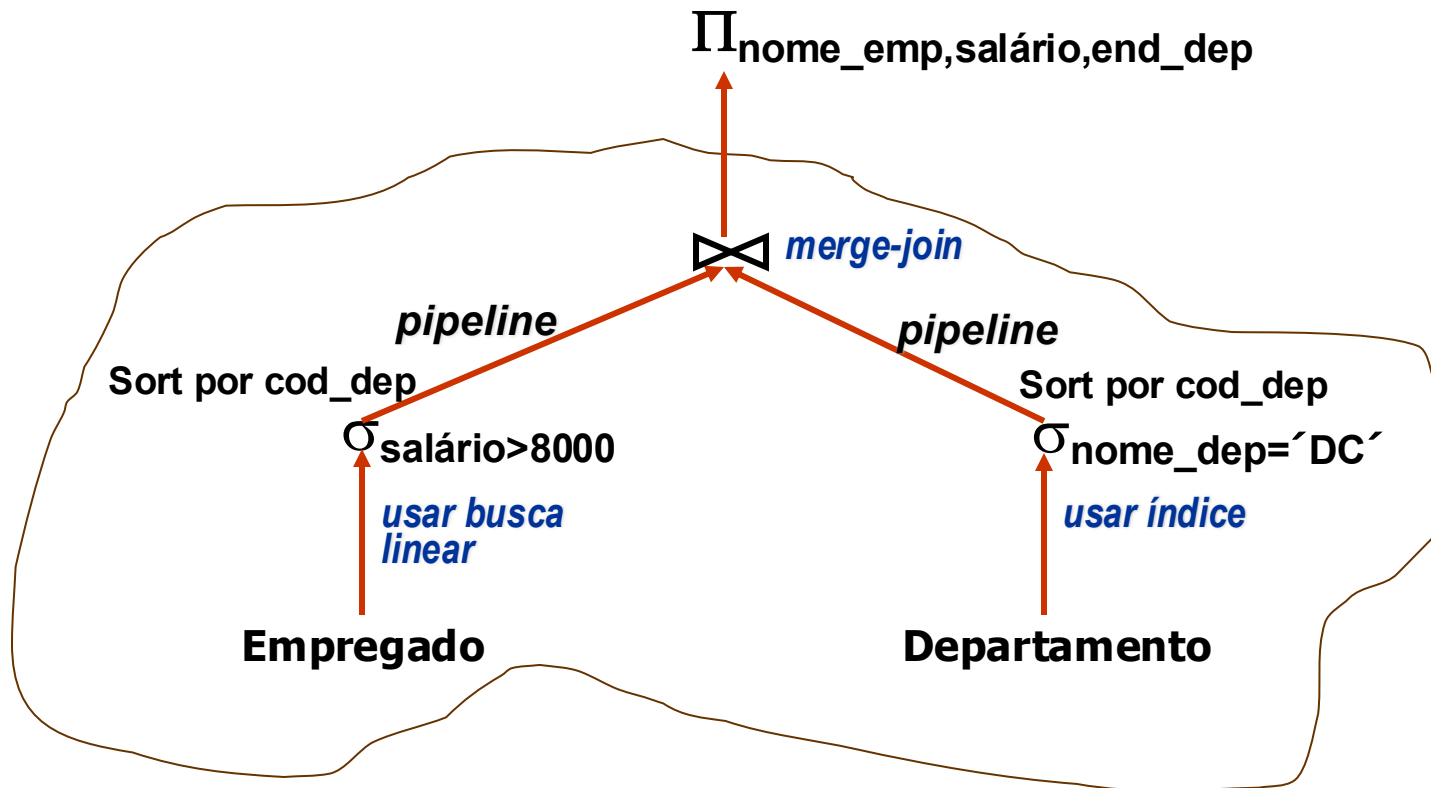
- Plano de Execução de Expressões (consultas)
  - Estratégia 1
    1. Ler o grafo de consulta de baixo para cima
    2. Para cada operação
      - 2.1 Executar
      - 2.2 Materializar o resultado em tabelas temporárias
  - Tabelas temporárias geralmente precisam ser armazenadas em disco
  - Alternativa para materialização de resultados intermediários
    - Executar várias operações simultaneamente em um *pipeline*
    - Dentro do *pipeline*
      - O resultado da execução de cada operação é passado para a próxima operação

- Plano de Execução de Expressões (consultas)
  - Estratégia 2



- Considerando que as duas operações de seleção geram suas saídas classificadas pelo atributo de junção e que a operação de junção é implementada por um merge-join

### □ Extensão do Plano de Acesso



# Processamento de Consultas

## - Cenários Diferentes -

---

- Processamento de Consultas em BDs Distribuídos
  - Custo de comunicação
  - Ganhos em performance ao executar em paralelo partes de uma consulta em diferentes sites
  
- Processamento de Consultas em BDs Orientado a Objetos
  - Diferente modelo
    - Diferentes construtores de tipo  $\Rightarrow$  diferentes operadores
    - Falta de um formalismo padrão  $\Rightarrow$  compromete a otimização