UFRPE

UAST
Unidade Acadêmica de Serra Talhada - PE
Desde 2006

BSi
Bacharelado em Sistemas de Informação

[Projeto de Banco de Dados]
Aula 05: Views, Índices, Stored Procedures, Triggers

Prof.: Msc. Hidelberg O. Albuquerque

Período Letivo: 2017.1

Visões

- São tabelas derivadas de outras tabelas;
- Permitem controlar a visão que cada usuário ou grupo de usuário tem sobre os dados armazenados;
  - Nem sempre é bom os usuários terem acesso a todos os dados do banco de dados;
- São tabelas virtuais, não são armazenadas fisicamente;
- Boas para representar consultas que devem ser feitas com muita frequência;

Visões

- Cada visão tem um nome, um conjunto de atributos e uma consulta que a representa;
- As visões contém sempre os dados atualizados;
  - A consulta é executada sempre que a visão é invocada
    - Garante dados sempre atualizados
      - Responsabilidade do SGBD
- A definição da visão é armazenada no catálogo do SGBD;
- Usamos a seguinte sintaxe para descrever uma visão:

CREATE VIEW *Nome*

AS *Consulta*

Visões

- Exemplo: Crie uma visão que ao ser invocada possa retornar os nomes dos empregados, o nome dos projetos e o número de horas trabalhadas em cada projeto.
  - Funcionario(id, matricula, nome, salario, id_depto)
  - Projeto(id, nome, id_depto, *#id_supervisor*)
  - Trabalha_Projeto(*#id_funcionario*, *#id_projeto*, numhoras)

```
CREATE VIEW funcionario_com_projetos
AS SELECT f.nome AS funcionario,
          p.nome AS projeto, tp.num_horas
FROM funcionario f, projeto p, trabalha_projeto tp
WHERE f.id=tp.id_funcionario AND
      p.id=tp.id_projeto
```

Visões

- Podemos fazer consultas em cima dos dados de uma visão;
  - Exemplo: Recupere o nome de todos os empregados que trabalham no projeto ‘Max Lucro’;

```
SELECT funcionario
FROM funcionario_com_projetos
WHERE projeto = 'consulta'
```

Visões

- Atualização:
  - Muitas vezes, os usuários podem querer inserir e atualizar dados em visões:
  - Isto pode trazer problemas:
    - Ex.: O que aconteceria se o usuário tentasse executar o comando?

```
INSERT INTO  funcionario_com_projetos
VALUES ('Kaique','Boa Oferta', '12')
```

Visões

- Atualização:
  - Em geral, a atualização de visões é tratada da seguinte forma:
    - Uma visão definida **numa única tabela** é atualizável SE os atributos da visão contêm a chave primária e todos os atributos que possuem a restrição NOT NULL sem nenhum valor default;
    - Visões definidas sobre **múltiplas tabelas usando junção** geralmente NÃO são atualizáveis;
    - Visões usando **funções de agrupamento** e **funções agregadas** NÃO são atualizáveis;

Visões

- Redefinindo uma visão:
  - Podemos redefinir o nome de uma visão através do operador ALTER VIEW;
    - Sintaxe:

      ALTER VIEW NomeDaVisão

      RENAME TO NovoNomeDaVisão;
    - Exemplo:

      ALTER VIEW funcionario_com_projetos

      RENAME TO funcionarios_com_projetos

Visões

- Excluindo uma visão:
  - Podemos excluir uma visão usando o operador DROP VIEW;
    - Sintaxe:
      - DROP VIEW NomeDaVisão
    - Exemplo:
      - DROP VIEW funcionario_com_projetos

Índices

- Um índice é uma estrutura de acesso físico aos dados, usado para acelerar o tempo das consultas;
  - Semelhante ao índice de um livro que usamos para encontrar um determinado assunto;
  - Definido sobre o valor de um atributo;
    - Geralmente, a chave primária e os atributos com a restrição UNIQUE;
  - Uma tabela pode ter mais de um atributo indexado

Índices

- Criando um novo índice:
  - Podemos criar um novo índice através do operador CREATE INDEX;
    - Sintaxe:

      CREATE INDEX NomeDoÍndice

      ON NomeDaTabela(Atributo)
    - Exemplo:

      CREATE INDEX ind1

      ON Empregado(Matricula)

Índices

- Especificando índices:
  - Existem vários tipos de manipulação de índices, baseados em estruturas de dados conhecidas.
    - Árvore B, Tabela Hash, entre outros.
  - Para especificar o tipo do índice, basta especificá-lo no momento de sua criação:
    - Sintaxe:

      CREATE INDEX NomeDoÍndice
      ON NomeDaTabela
          USING Nome_Estrutura(Atributo)
    - Exemplo:

      CREATE INDEX ind1
      ON Empregado USING btree(Matricula) /*Árvore B*/

Índices

- Especificando índices:
  - Cada SGBD utiliza os algoritmos específicos.
    - Consulta à API do SGBD.
  - No PostgreSQL, não é necessário especificar o método btree(), porque é o padrão da linguagem para a criação de índices.

Índices

- Alterando um índice:
  - Podemos alterar um índice existente através do operador ALTER INDEX;
    - Sintaxe:

      ALTER INDEX NomeDoÍndice
      RENAME TO NomeDoNovoÍndice
    - Exemplo:

      ALTER INDEX indiceSalario
      RENAME TO indiceSalarioRenomeado

Índices

- Removendo um índice:
  - Podemos remover um índice usando o operador DROP INDEX;
    - Sintaxe:

      DROP INDEX NomeDoIndice;
    - Exemplo:

      DROP INDEX indiceSalarioRenomeado;

Índices e Árvores B

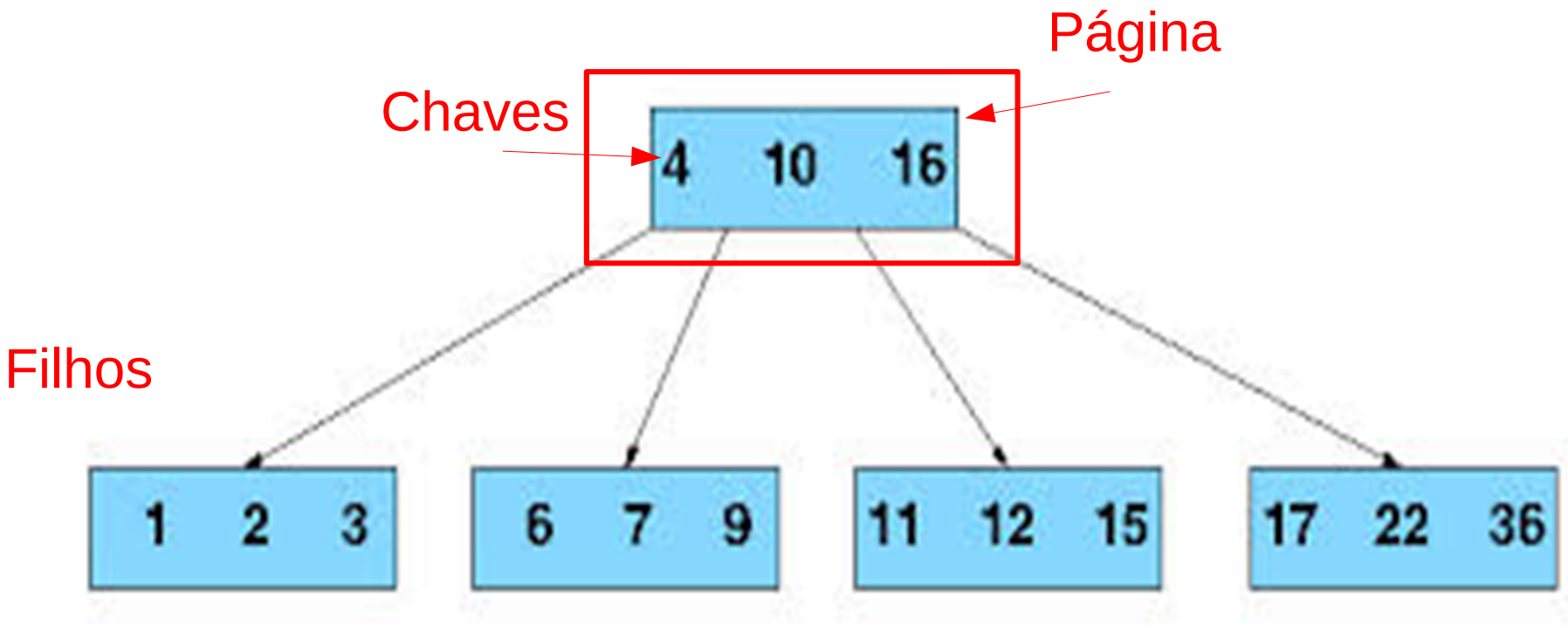
- Árvore de pesquisa balanceada, utilizada para acessar memória secundária.
  - Memória Primária: RAM. Memória Secundária: Discos.
  - É a forma como o SO acessa os dados no HD.

- Diferente da árvore binária, as árvores B são multidimensionais.
  - Árvores Binárias possuem no máximo 2 filhos e 1 chave.
  - Árvores B possuem N filhos e N-1 chaves.

- Muitos SGBDs utilizam Árvore B ou alguma variante.
  - Árvores B+

Árvores B

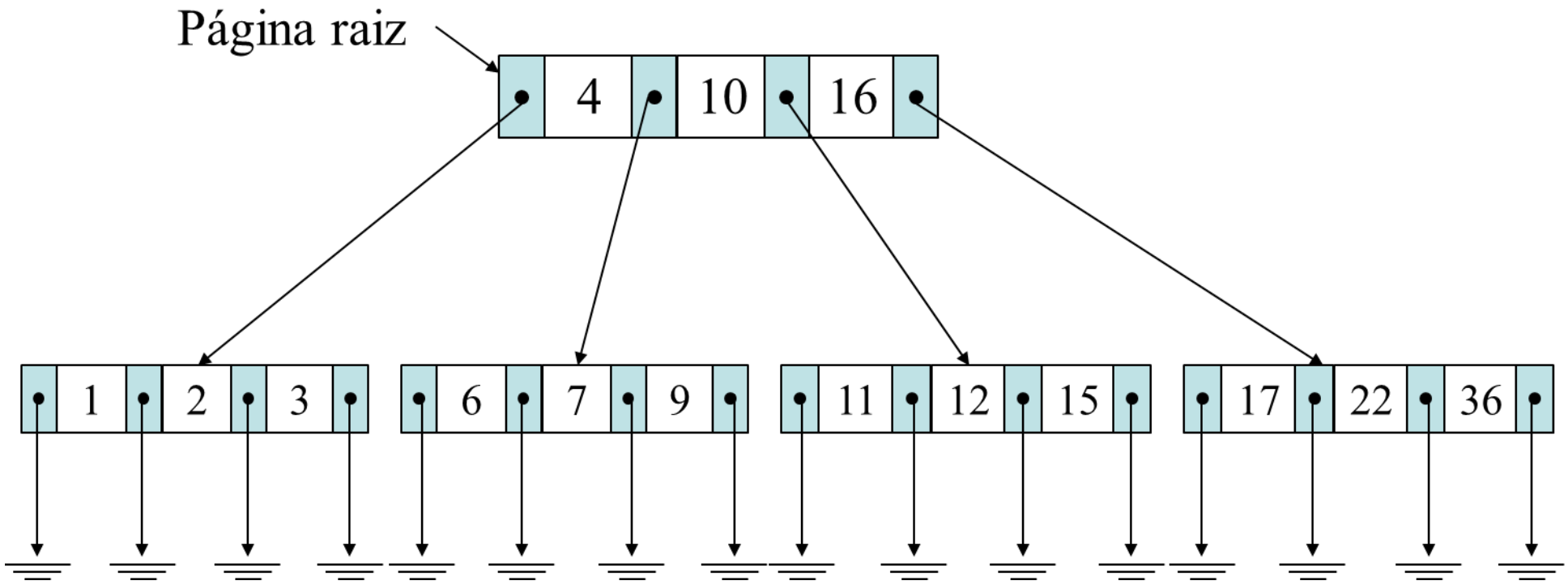
- Visão Geral:



  - Obs:
    - Conjunto de 1 ou + chaves = página (em binária seria nó)
      - Alguns autores utilizam também o nome “Nó”
    - Página Raiz, Página Folhas, Páginas Internas.

Árvores B

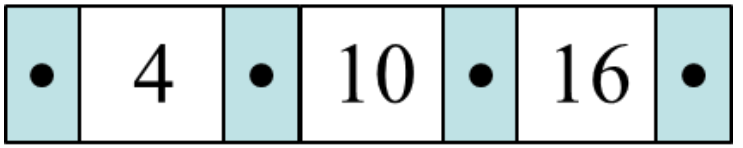
- Representação na Memória:



  - As árvores B são uma generalização no uso da árvores binárias
  - A raiz sempre estará na memória principal
    - Se um dado não está na raiz, ele desce para o disco.
    - Rapidez

Árvores B

- Propriedades (1):
  - Uma página deve conter:



    - Número “X” de chaves atualmente armazenado
    - As próprias chaves (valores)
      - Chaves tem que estar na ordem **crescente**
    - Atributo booleano “folha”
    - Número de ponteiros “X+1”
      - X = quantidade de chaves
        - Sempre haverá mais ponteiros que chaves.
      - Se não tem filhos, apontam pra null

Árvores B

- Propriedades (2):
  - Todas as folhas estão no mesmo nível da árvore
    - O nível das folhas é a profundidade da árvore
      - Em árvore binária, profundidade ou altura é a mesma coisa
  - Grau mínimo “t”
    - Quantidade mínima de filhos (ou ponteiros) de uma página.
    - Toda árvore deve ter t >= 2.
  - Limites:
    - Indica a quantidade de chaves possíveis numa página.
      - Limite Inferior: t-1
      - Limite superior: 2t-1
  - Exceção: raiz pode ter mínimo 1 chave, independente de t.

Árvores B

- Inserção de Elementos:
  - Toda inserção é feita numa folha.
    - Os limites inferior e superior têm que ser respeitados.
  - Antes de inserir o elemento, percorre a árvore para encontrar posição a ser inserida.
    - Caso 1: Se página for uma folha, e puder receber dado (se tiver espaço), insere.
    - Caso 2: Se a página for uma folha e estiver completa (cheia):
      - A página-folha tem que ser dividida ao meio (antes da inserção)
      - O indivíduo do meio é promovido para a raiz. Os elementos menores que ele serão filhos esquerdos, e os maiores, filhos direitos.
      - Se a raiz que receberá o novo elemento estiver completa, o processo de divisão também se aplica recursivamente pra cima, até balancear a árvore.
      - Após a divisão, insere o elemento no seu devido local.

Árvores B

- Inserção de Elementos:
  - Perceba que o crescimento das árvores B é de baixo para cima.
    - Binárias crescem pra baixo.
  - As inserções devem ser feitas numa única passagem
    - OU seja: divide antes de inserir, e não insere pra depois dividir

Árvores B

- Inserção de Elementos:
  - Exemplo: * Construir uma árvore B, com t=2
    * Limites Chaves: inferior: t-1 (1) e Limite superior: 2t-1 (3)
    * cada página vai ter: se 1 chave (inferior), 2 filhos;
      se 2 chaves, 3 filhos
      se 3 chaves (superior), 4 filhos
  - Chaves a serem inseridas: **U**EQACPODW
  - 1ª Inserção: U

    - como não tem raiz, cria e insere
    - nestes exemplos, só iremos mostrar os ponteiros para filho qdo houverem.
    - nestes exemplos, iremos mostrar os elementos da página qdo necessário.

Árvores B

- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: U**E**QACPODW
  - 2ª Inserção: E
    - Como a raiz é uma folha e tem espaço (caso 1) insere ordenado na raiz

Árvores B

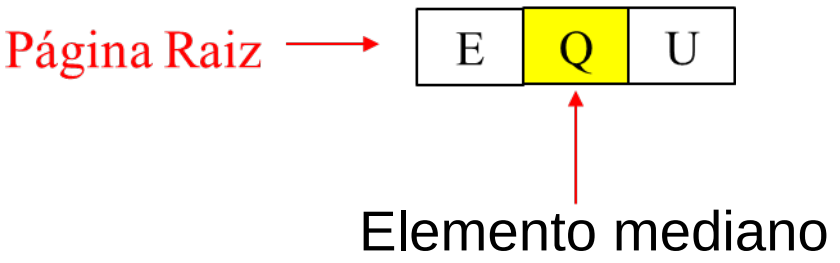
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UE**Q**ACPODW
  - 3ª Inserção: Q
    - Como a raiz é uma folha e tem espaço insere ordenado na raiz

Árvores B

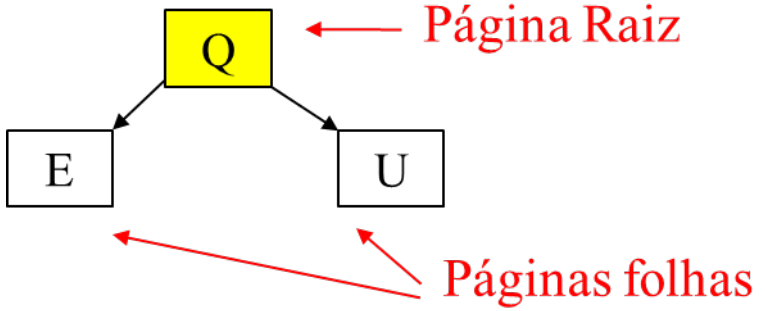
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQ**A**CPODW

  - 4ª Inserção: A
    - Como a raiz-folha está completa (Caso 2) é preciso
      1) dividir a página:
      2) o nó mediano é promovido para a raiz.
      3) demais elementos são alocados mediante sua posição na página

Antes da Divisão:



Depois da Divisão:

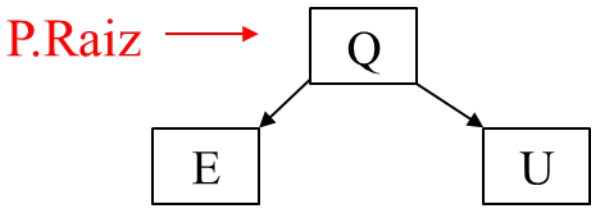


Árvores B

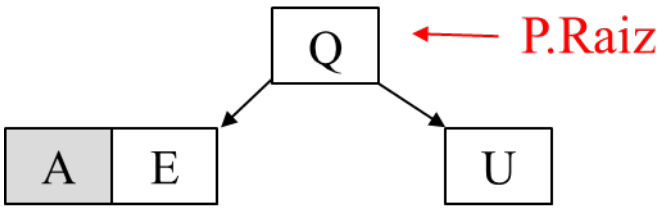
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQ**A**CPODW

  - 4ª Inserção (continuação): A
    - Após a divisão da página, insere o elemento na folha apropriada.

Antes da Inserção:



Depois da Inserção:

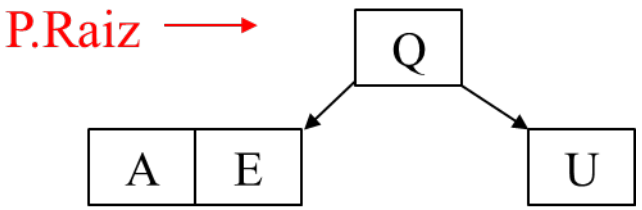


Árvores B

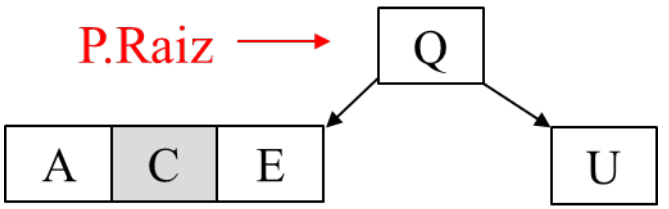
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQA**C**PODW

  - 5ª Inserção: C
    - Localiza a folha-posição a ser inserida, verifica se está cheia ou não.
    - Aplica o caso correspondente

Antes da Inserção:



Depois da Inserção:

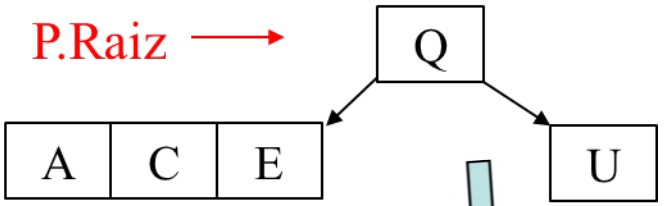


Árvores B

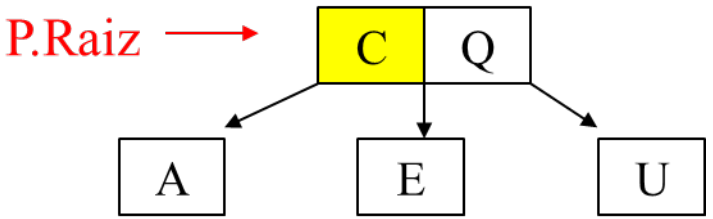
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQAC**P**ODW

  - 6ª Inserção: P
    - Localiza a folha-posição a ser inserida, verifica se está cheia ou não.
    - Aplica o caso correspondente (Caso 1 ou Caso 2)

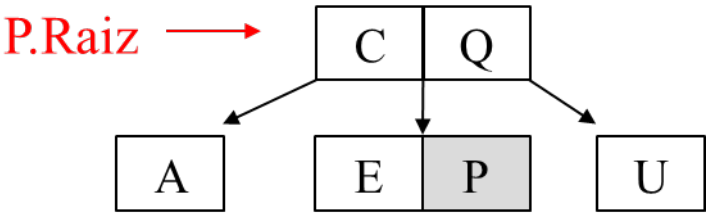
a) Antes da Inserção:



b) Faz a Divisão



c) Insere na folha

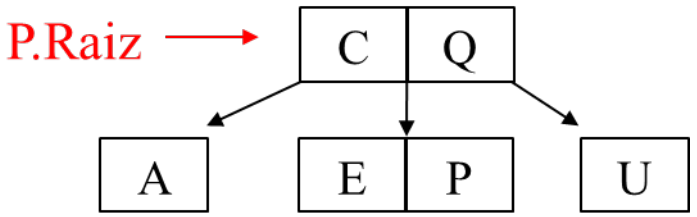


Árvores B

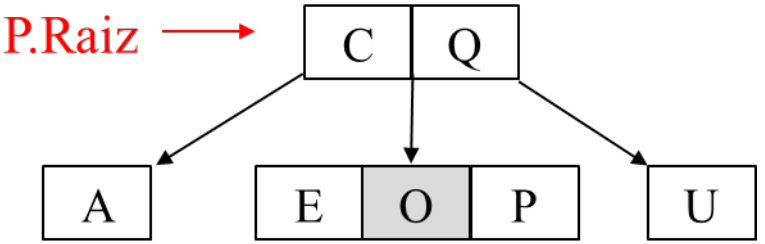
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQACP**O**DW

  - 7ª Inserção: O
    - Localiza a folha-posição a ser inserida, verifica se está cheia ou não.
    - Aplica o caso correspondente (Caso 1 ou Caso 2)

a) Antes da Inserção:



b) Depois da Inserção

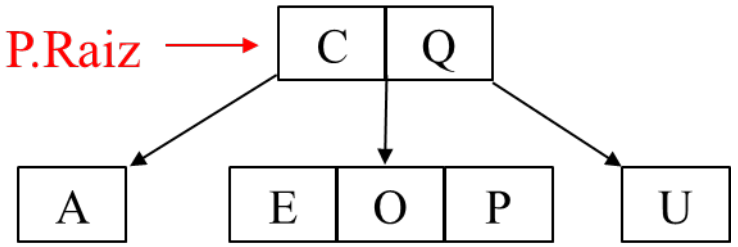


Árvores B

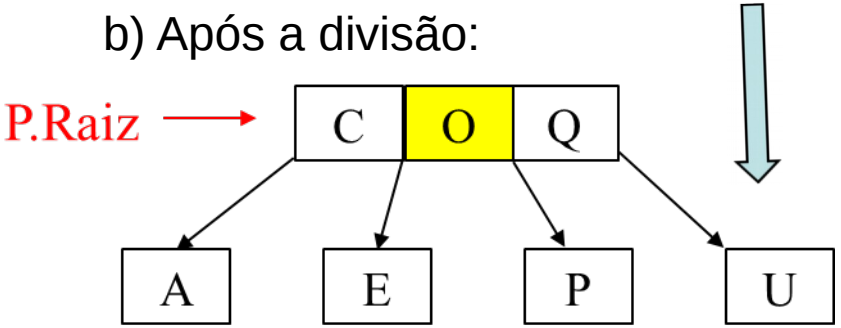
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQACPO**D**W

  - 8ª Inserção: D
    - Localiza a folha-posição a ser inserida, verifica se está cheia ou não.
    - Aplica o caso correspondente (Caso 1 ou Caso 2)

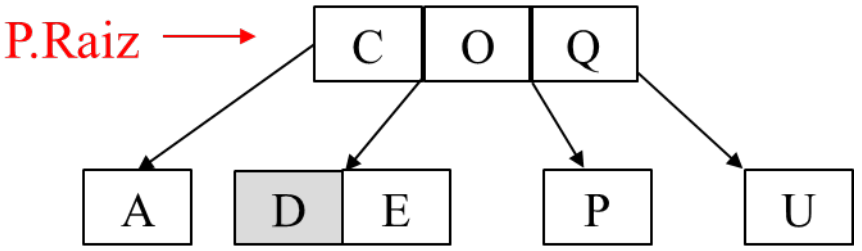
a) Antes da Inserção:



b) Após a divisão:



c) Depois da Inserção

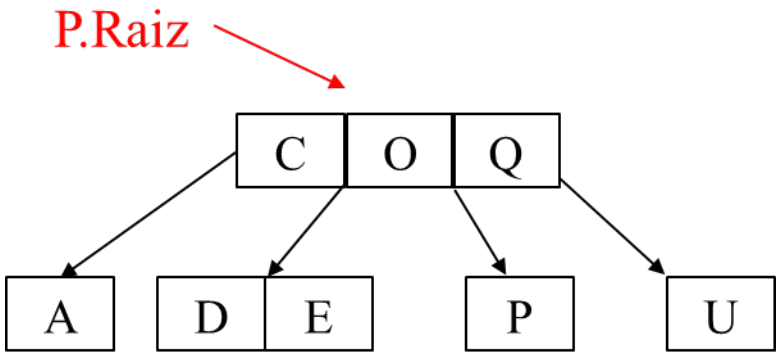


Árvores B

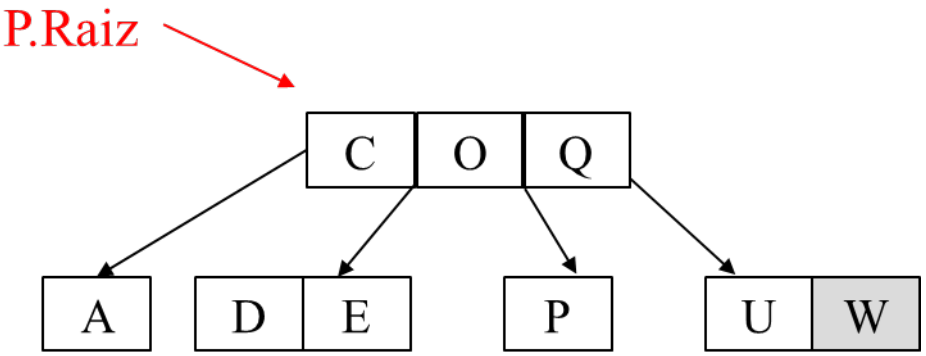
- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQACPOD**W**

  - 9ª Inserção: W
    - Localiza a folha-posição a ser inserida, verifica se está cheia ou não.
    - Aplica o caso correspondente (Caso 1 ou Caso 2)

a) Antes da Inserção:



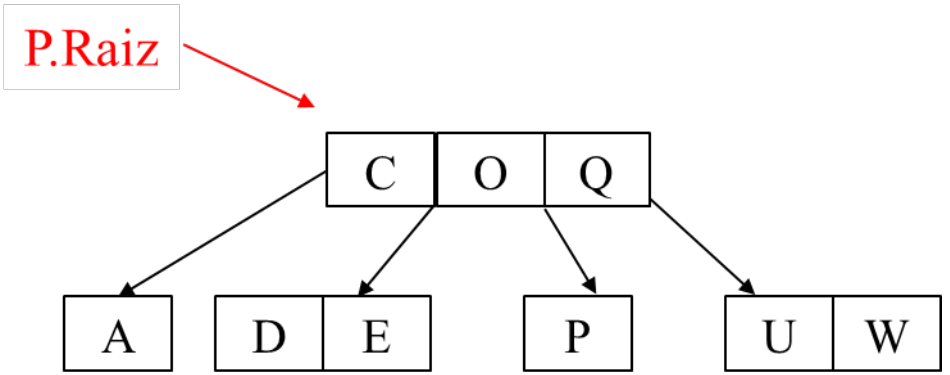
c) Depois da Inserção



Árvores B

- Inserção de Elementos:
  - Exemplo: * t=2, LI = 1 e LS = 3
  - Chaves a serem inseridas: UEQACPODW

  - ÁRVORE B FINAL



  - Segundo CORMEN,
  - “A árvore B mais simples ocorre quando t = 2. Todo nó interno tem 2, 3 ou 4 filhos, e temos uma árvore 2-3-4. Na prática, porém, em geral são utilizados valores de t muito maiores.”

  - (CORMEM, T. H. et al. Algoritmos, Teoria e Prática. Tradução da 2ª edição americana. Rio de Janeiro: Campus, 2002).

Árvores B

- Inserção
  - Toda operação de inserção/remoção deve respeitar os limites inferiores e superiores.
  - Alguns aplicativos utilizam uma forma diferente de inserção:
    - Quando a página a ser inserida está cheia, 1) insere pra depois dividir e promover.
  - Esta abordagem acima não é recomendada, porque é mais custosa.
    - “Dividir” uma página e “promover” o elemento mediano antes da inserção proporciona as operações em uma única passagem na árvore.
      - Esta é a única abordagem que será aceita na disciplina.

Árvores B

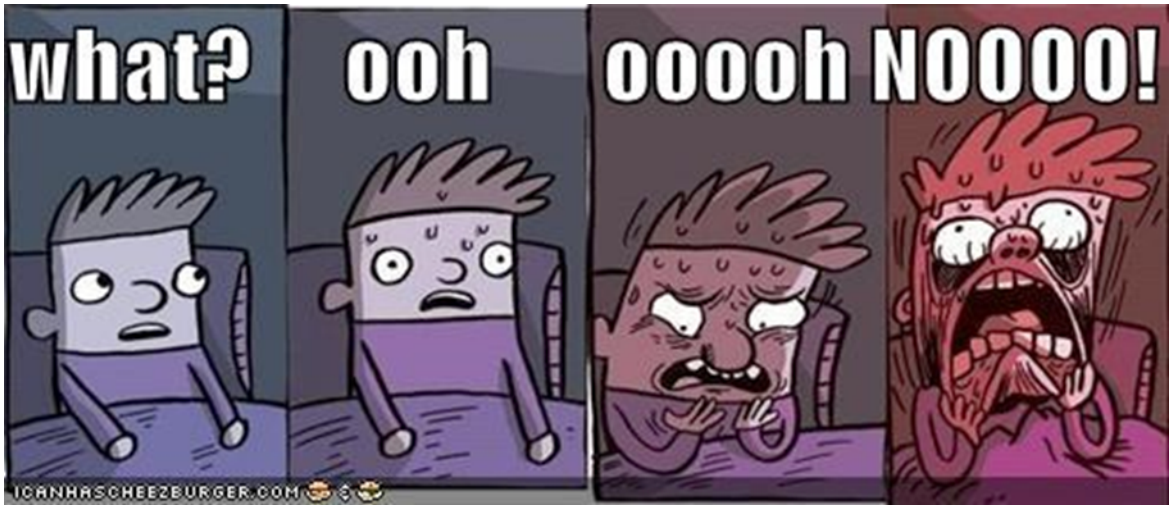
- Exercício:
  - Analise a inserção dos elementos {26, 20, 24, 36, 78, 87, 38, 55, 54, 44, 12, 79, 32, 67, 97, 48, 90, 10, 85, 6, 80, 92, 1}, na ordem apresentada.
  - A árvore terá t=3, e Limite Inferior (t-1) = 2 e Limite Superior (2t-1): 5
  - Quantidade de filhos por página: [qtde de chaves] + 1
  - Não serão apresentados passo a passo. Tente fazer no papel e validar sua resposta.

Árvores B

- Busca:
  - Similar à busca binária.
  - Recebe o valor a ser procurado e a raiz.
  - Procura o dado na raiz.
    - Se encontrar, retorna o valor.
    - CC, retorna nulo.

Árvores B

- Remoção
  - Existem 6 casos possíveis.



  - O detalhamento dos 6 casos ficará como atividade de pesquisa para o aluno.
  - O professor estará à disposição para retirar possíveis dúvidas.

Árvores B

- Sobre Índices e Árvores B no PostgreSQL:
  - https://www.postgresql.org/docs/9.3/static/sql-createindex.html
  - http://ieftimov.com/postgresql-indexes-btree
  - http://www.devmedia.com.br/trabalhando-com-indices-no-postgresql/34028

Integridade de Banco de Dados

- Alguns recursos podem ser usados para manter a integridade do banco de dados:
  - Restrições de tuplas;
  - Asserções;
  - Procedimentos Armazenados;
  - Gatilhos;

  - Próximas aulas...

Restrições de Tuplas

- Podemos especificar restrições de tuplas usando o comando CHECK no momento da criação de uma tabela;
- Exemplo:

```
CREATE TABLE funcionario(
    matricula VARCHAR(14),
    nome VARCHAR(45) NOT NULL,
    salario REAL,
    CHECK (salario>350),
    PRIMARY KEY (matricula)
)
```

Restrições de Tuplas

- Restrições de tuplas são **restrições a nível de uma única tupla**, ou seja, restrições que ***não precisam*** de comparações com outras tuplas.
- Inviável para especificar restrições mais complexas:
  - Ex.: Nenhum funcionário pode ganhar mais que seu supervisor.
- Para estes casos, Asserções se tornam uma solução mais prática.

Asserções

- Tipos adicionais de restrições que estão fora do escopo das restrições “padrão” do modelo relacional
  - Chave Primária e Unicidade
  - Integridade de identidade (check)
  - Integridade referencial (chave estrangeira)
- As asserções são restrições genéricas, que deve ser definida na criação da tabela (mas não somente):

  CREATE ASSERTION nome_da_assercao

  CHECK (restricao_integrada)

- Geralmente, ao definir uma asserção, selecionamos as possíveis tuplas e usamos a clausula **NOT EXISTS** para verificar que o conjunto recuperado é vazio;
- Mas outras formas também podem ser usadas;

Asserções

- Exemplo: Vamos supor a seguinte restrição de integridade: “Nenhum empregado pode ganhar mais que seu supervisor”.

```
CREATE ASSERTION salario_menor
CHECK
(NOT EXISTS (
    SELECT * FROM empregado e, empregado s
    WHERE e.supervisor=s.matricula AND
          e.salario>s.salario
    )
)
```

Asserções

- Para especificar esta restrição através de uma asserção:
  - Especificamos uma consulta para recuperar os empregados que ganham mais do que o seu supervisor;
  - Usamos a cláusula NOT EXISTS para assegurar que nenhuma tupla foi recuperada;
  - Caso o resultado de consulta não seja vazio, a restrição de integridade foi violada;

Asserções

- O acontece quando definimos uma asserção?
  - Sempre que alguma das tabelas for alterada, a asserção é verificada pelo SGBD;
  - Atualizações que violam a restrição (a asserção) são automaticamente rejeitadas pelo SGBD;

- Asserções provocam um grande overhead ao SGBD;
  - Principalmente quando muitos usuários podem atualizar o banco de dados simultaneamente;
  - Muitos SGBD's não oferecem suporte a asserções:
    - Como SQLServer e PostgreSQL;

Stored Procedures

- São subprogramas escritos para realizar uma tarefa especifica;
- Semelhante ao conceito de subprogramas em linguagens de programação;
- São armazenados de forma persistente no cátalogo do SGBD;
- Embora conhecidos como *Stored Procedure*, podem ser procedimentos ou funções;
- Podem ser executados pelo SGBD no momento de uma consulta;

Stored Procedures

- Os procedimentos armazenados oferecem as seguintes vantagens:
  - Criar subprogramas no próprio SGBD diminui a complexidade da aplicação e facilita o trabalho de atualização e manutenção;
    - É mais fácil alterar o subprograma no SGBD do que o código de todas as aplicações que acessam os dados;
  - Diminuem o tráfego de dados na rede;
    - Apenas os dados já processados circulam na rede;

Stored Procedures

- Stored Procedures utilizam a linguagem a linguagem PL/SQL
  - Procedural Language extensions to SQL
- Torna possível a construção de aplicações eficientes para a manipulação grandes volumes de dados (tabelas com milhões ou bilhões de registros).
- A eficiência da PL/SQL também é garantida através da sua forte integração com a linguagem SQL.
  - É possível executar comandos SQL diretamente de um programa PL/SQL, sem a necessidade da utilização de API's intermediárias (como ODBC ou JDBC).
- Geralmente, muitos SGBD's preferem desenvolver sua própria linguagem

Stored Procedures

- A Linguagem PL/pgSQL
  - Procedural Language/PostgreSQL Structured Query Language
- É a linguagem usada pelo SGBD PostgreSQL para a definição de procedimentos armazenados;
- A linguagem SQL/PSM – (SQL/Persistent Stored Methods) é o padrão SQL para a especificação de storeds procedures;

Stored Procedures

- Sintaxe para a criação de funções:

```
CREATE FUNCTION nome_da_funcao(parametro(s))
RETURNS  tipo_de_retorno
AS '
      DECLARE
            Declaracaoes_locais;
      BEGIN
            Corpo_da_funcao;
            RETURN  resultado;
      END '
LANGUAGE linguagem;
```

Stored Procedures

- Declaração de parâmetros:
  - Cada procedimento armazenado pode ter zero ou mais parâmetros de entrada;
  - O SGBD define os parâmetros de entrada com os nomes $1, $2, $3,..., $n, onde:
    - $1 é o primeiro parâmetro, $2 é o segundo, .. $n é o último

- Desta forma, declaramos apenas os tipos dos parâmetros de entrada;

Stored Procedures

- Declaração de parâmetros:
  - Exemplo:

    CREATE FUNCTION exemplo (VARCHAR, INTEGER)

    RETURNS INTEGER

    (...)

  - Automaticamente o SGBD entende que a função “exemplo” tem os parâmetros;
    - $1, do tipo VARCHAR;
    - $2, do tipo INTEGER;

Stored Procedures

- Declaração de parâmetros:
  - No entanto, para fins de legibilidade, podemos redefinir o nome de um parâmetro na seção DECLARE;
  - Para isso, usamos a seguinte sintaxe, por exemplo:
    - NovoNome ALIAS FOR $1;
  - Exemplo:

```
CREATE FUNCTION exemplo (VARCHAR, INTEGER)
RETURNS INTEGER
AS '
DECLARE
        Parametro1 ALIAS FOR $1;
        Parametro2 ALIAS FOR $2;
        ....
```

Stored Procedures

- Tipos de Retorno
  - PL/SQL permite que uma função retorne qualquer tipo ou array de um tipo de dado oferecido pelo servidor;
  - Outros tipos mais complexos, como registros, conjuntos, tabelas e tipos definidos pelo usuário também são permitidos;
  - O tipo **VOID** é usado para indicar que nenhum valor será retornado;

Stored Procedures

- Declaração de Variáveis Locais
  - Todas as variáveis do subprograma devem ser declaradas na seção DECLARE;
    - Nesta seção também podemos declarar constantes;
  - A declaração de uma variável deve obedecer à seguinte sintaxe:

nomeDaVariavel [CONSTANT] Tipo [NOT NULL] [{DEFAULT |:=}] [Expressão]

Stored Procedures

- Declaração de Variáveis Locais
  - Onde:
    - nomeDaVariavel: define como a variável será identificada dentro do subprograma;
    - CONSTANT: Palavra chave que define que o identificador é uma constante;
    - Tipo: Define o tipo de valor que será armazenado na variável;
    - NOT NULL
      - Identifica que a variável não pode assumir um valor nulo;
      - Caso ele seja usada, um valor default deve ser especificada para a mesma;

Stored Procedures

- Declaração de Variáveis Locais
  - Onde:
    - DEFAULT
      - Representa o valor default assumido pela variável;
      - É obrigado caso ela tenha a restrição NOT NULL;
    - :=
      - Usado para atribuir um valor inicial para a variável;
    - Expressão:
      - Expressão usada para atribuir um valor à variável;
      - O resultado da expressão deve ser compatível com o tipo da variável;

Stored Procedures

- Declaração de Variáveis Locais
  - Exemplos de declaração de variáveis locais:
    - DECLARE

```
quantidade INTEGER;
valor NUMERIC(2) DEFAULT 200;
soma INTEGER := 0;
pi CONSTANT NUMERIC(2) := 3.14;
idade INTEGER NOT NULL DEFAULT 0;
sexo VARCHAR(1);
```

Stored Procedures

- Atribuindo valor a uma variável
  - Podemos atribuir valor a uma variável da seguinte forma:

    nome_da_variavel := expressão;

    - Expressão é um valor **OU** uma expressão compatível como tipo da variável;

  - Exemplos:

    quantidade := 0;

    cont := cont + 1;

Stored Procedures

- Atribuindo valor a uma variável
  - Podemos também atribuir a uma ou mais variáveis o resultado de uma consulta;
  - Para isso, incluímos uma cláusula SELECT INTO seguida das variáveis e depois as colunas recuperadas na consulta antes da cláusula FROM da consulta, da seguinte forma:

```
SELECT INTO V1,V2,....,Vn  C1,C2,....,Cn
FROM .....;
```

Stored Procedures

- Atribuindo valor a uma variável

  SELECT INTO V1,V2,....,Vn C1,C2,....,Cn
  FROM .....;

  - Onde:
    - Cada Vi corresponde a cada variável que receberá o valor de uma das colunas recuperadas na consulta;
    - Cada Ci corresponde a cada coluna recuperada na consulta;
      - Aquelas que usamos antes da cláusula FROM;
    - O número de colunas recuperadas deve ser compatível com o número de variáveis usadas;
      - E os tipos também devem ser compatíveis com suas respectivas variáveis;

Stored Procedures

- Atribuindo valor a uma variável
  - Exemplos:

```
SELECT INTO salarioAtual   Salario
FROM Empregado
WHERE matricula = '1111-1';

SELECT INTO total, media   COUNT(*), AVG(Salario)
FROM Empregado;

SELECT INTO emp *
FROM Empregado
WHERE matricula='1111-1';
```

    - Em relação ao último exemplo é perfeitamente possível, caso a variável “emp” tenha sido declarada com um RowType da tabela Empregado;
      - Veremos mais adiante sobre o RowType

Stored Procedures

- Exemplo 1: Criar um subprograma para recuperar o próximo código de USUARIO a ser inserido:

```
CREATE OR REPLACE FUNCTION proximoCodigoUsuario()
RETURNS INTEGER
AS '
    DECLARE
            maiorCodigo  INTEGER;
     BEGIN
            SELECT INTO  maiorCodigo MAX(id)
            FROM usuario;
            RETURN  maiorCodigo+1;
      END '
LANGUAGE plpgsql;
```

Stored Procedures

- Note que, diferentemente do que vimos em SQL, a constante que corresponde ao nome do empregado está entre aspas duplas;
  - Isto acontece porque o corpo do subprograma é definido entre apóstrofos;
  - O SGBD pode confundir o apóstrofo do string como o fechamento do corpo da função;
  - Para que isso não aconteça, usamos um apóstrofo de escape para cada apóstrofo de usados em string ou caractere;

Stored Procedures

- **Exemplo 2**: Criar um subprograma que recupere a quantidade de projetos em que um determinado empregado trabalha:

```
CREATE OR REPLACE FUNCTION qtdeProjetosEmp(VARCHAR)
RETURNS INTEGER
AS '
    DECLARE
        id_emp ALIAS FOR $1;
        total INTEGER;
    BEGIN
        SELECT INTO  total COUNT(*) FROM Trabalha_Projeto TP
        WHERE tp.empregado=id_emp;
        RETURN  total;
    END '
LANGUAGE plpgsql;
```

Stored Procedures

- Para executar um procedimento armazenado a partir de uma consulta SQL;
  - Podemos executá-los também em uma consulta interna a outro subprograma;
  - Exemplos:
    - SELECT proximoDepartamento();
    - SELECT qtdeProjetosEmp('1111-1');
- Podemos chamar um procedimento armazenado a partir do resultado de uma consulta;
  - Exemplo: Recuperar o codigo e o nome de cada empregado com a quantidade de projetos em que ele trabalha;
    - SELECT id, nome, qtdeProjetosEmp(id) AS numeroProjetos FROM empregado

Stored Procedures

- Podemos também usar o resultado de um procedimento na cláusula WHERE;
  - Exemplo: Recuperar a matricula e o nome dos empregados que trabalham em mais de um projeto;
    - SELECT id, nome FROM empregado WHERE qtdeProjetosEmp(id)>1

Stored Procedures

- Estruturas de controle: o comando IF-THEN-ELSE:
  - Permite a especificação de desvios condicionais;
  - Semelhante ao comando IF de uma linguagem de programação;
  - A cláusula ELSE é opcional;
  - A sintaxe do comando é a seguinte:

```
IF condição THEN comandos;
    ELSIF condição THEN comandos;
        ELSE comandos;
END IF;
```

Stored Procedures

- Estruturas de controle: o comando IF-THEN-ELSE:

```
IF condição THEN comandos;
        ELSIF condição THEN comandos;
          ELSE comandos;
END IF;
```

  - Onde:
    - *Condição* é uma expressão lógica;
    - *Comandos* representam uma sequência de comandos aceitos definidos na linguagem PL/SQL;
    - As cláusulas IF, ELSIF e ELSE podem executar qualquer quantidade de comandos;
      - Sem a necessidade de usar um bloco BEGIN-END;

Stored Procedures

- Exemplo: Criar um subprograma que classifique um empregado de acordo com a quantidade de telefones que ele tem. Considerando que se ele tiver somente 1 telefone, o programa informe “fala pouco”; se ele possui entre 2 ou 3 telefones, “fala razoavel”; se possuir mais de 3 telefones, “fala muito”.

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - PL/SQL oferece as seguintes estruturas de repetição:
    - LOOP
    - WHILE
    - FOR
    - As palavras-chaves EXIT e CONTINUE ajudam na definição destes laços;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **EXIT:**
      - Usada para indicar que um determinado laço deve ser encerrado;
      - Ela pode ser usada com um label e uma condição;
        - Mas, ambos, podem ser omitidos;
      - Caso um label seja especificado, ela vai encerrar o comando especificado pelo label;
        - Caso contrário, o loop mais interno será encerrado;
      - Caso uma condição seja especificada, o encerramento só acontecerá se a condição for satisfeita;
        - Caso contrário o loop é encerrado imediatamente;
      -
      - A sintaxe para a especificação de uma condição de saída é:

EXIT label **WHEN** condição;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **CONTINUE**:
      - Usada para indicar que um determinado laço deve ser continuado;
      - A sua definição é semelhante à definição de uma condição de saída;
        - Ela também pode ser associada a uma condição e a um label;
      - A sua sintaxe é:

        CONTINUE label **WHEN** Condição;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **LOOP**:
      - LOOP é um comando que repete uma ação até que uma condição de encerramento seja executada;
      - Também podemos usar as cláusulas EXIT ou RETURN para encerrar o LOOP
      - A sua sintaxe é:

```
LOOP
    Comandos
END LOOP;
```

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **WHILE**:
      - Indica que um ou mais comandos devem ser repetidos enquanto uma determinada condição for satisfeita:
      - Semelhante a um comando WHILE de uma linguagem de programação;
      - A sua sintaxe é:

        WHILE Condição **LOOP**
        Comandos;
        END LOOP;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **FOR**:
      - Usado em situações onde sabemos antecipadamente o número de repetições;
      - A sua sintaxe é:
        - **FOR** variavel **IN [REVERSE]** incío..fim **[BY valor]**

```
LOOP
        Comandos
END LOOP;
```

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **FOR**:
      - **FOR** variavel **IN [REVERSE]** incío..fim **[BY valor]**

        LOOP

        Comandos

        END LOOP;
    - Onde:
      - *variável*:
        - É a variável de controle do comando FOR;
        - É através do seu valor que o SGBD sabe quando o laço deve ser encerrado;
        - Deve ser do tipo inteiro;
      - *IN*:
        - Usado para declarar o intervalo dos valores que serão assumidos pela variável de controle;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **FOR**:
      - **FOR** variavel **IN [REVERSE]** incío..fim **[BY valor]**

        LOOP

        Comandos

        END LOOP;
    - Onde:
      - *REVERSE*:
        - Indica que o intervalo de valores está na ordem decrescente (decremento)
      - *início..fim*:
        - Indicam o intervalo de valores que serão assumidos pela variável de controle;
        - Devem ser do tipo inteiro;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **FOR**:
      - **FOR** variavel **IN [REVERSE]** incío..fim **[BY valor]**

        LOOP

        Comandos

        END LOOP;
    - Onde:
      - BY:
        - Usado para informar em quanto a variável de controle deve ser incrementada ou decrementada após cada iteração;
        - O valor associado à mesma deve ser do tipo inteiro;
        - É uma cláusula opcional:
          - Se ela não é especificada, é considerado o valor 1;

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - **FOR**:
      - Exemplos:

```
FOR i IN 1..10 LOOP
        Comandos
END LOOP;

FOR i IN REVERSE 10..1 LOOP
        Comandos
END LOOP;

FOR i IN 1..10 BY 2 LOOP
        Comandos
END LOOP;
```

Stored Procedures

- Estruturas de controle:
  - Estruturas de repetição:
    - Podemos também usar o comando FOR para percorrer o resultado de uma consulta;
    - Neste caso, a variável de controle é do tipo “linha” (RowType);
      - Uma linha da tabela onde a consulta é realizada;
      - Também pode ser um registro compatível com os campos recuperados na consulta;
    - A cada iteração do comando, os valores da próxima tupla são atribuídos à variável de controle;
    -
    - Neste caso, usamos a seguinte sintaxe:

```
DECLARE
    variavel NomeTabela%RowType;
[...]
FOR variavel IN consulta LOOP
      Comandos;
END LOOP;
```

Stored Procedures

- O tipo **Cursor**:
  - Muita vezes, temos que percorrer o resultado de consultas para realizar alguma tarefa em um procedimento armazenado;
  - Para isso, temos que percorrer iterativamente cada tupla recuperada no resultado;
    - Com o comando FOR, este caminhamento é feito apenas em uma direção;
  - Uma variável do tipo cursor é usada para realizar esta tarefa;
  - É como se fosse um ponteiro que navegasse nas tuplas do resultado;

Stored Procedures

- O tipo **Cursor**:
  - Todo cursor é uma variável do tipo **REFCURSOR**;
  - Ele pode ser declarado com uma variável qualquer na seção DECLARE;
    - Neste caso, o cursor pode ser usado em qualquer consulta;
      - Ou até em mais de uma consulta no subprograma;
    - O cursor é chamado de não limitado quando ele não está associado apenas a uma consulta;
  - Exemplo:

```
DECLARE
        C1 REFCURSOR;
```

Stored Procedures

- O tipo **Cursor**:
  - Também podemos declarar um cursor através da seguinte sintaxe:

    nomeDoCursor REFCURSOR FOR Consulta;

  - Neste caso, o cursor é chamado de limitado, pois é associado a uma consulta especifica;
    - Exemplo:

      C1 REFCURSOR FOR SELECT * FROM Empregado;

  - Depois de realizada a consulta, precisamos abrir o cursor antes de usá-lo para navegar no resultado da consulta;
    - No caso de um cursor limitado, usamos a seguinte sintaxe:

      OPEN nomeCursor FOR Consulta
  - Exemplo:
    - OPEN c1 FOR SELECT * FROM TrabalhaProjeto;

Stored Procedures

- O tipo **Cursor**:
  - Quando o cursor é não limitado, ele não pode estar previamente aberto por causa de outra consulta;
  - Para o cursor não limitado, usamos a seguinte sintaxe:
    - OPEN nomeCursor;
  - Exemplo:

```
OPEN c1;
```

Stored Procedures

- O tipo **Cursor**:
  - Depois que o cursor é aberto, usamos a operação FETCH para obter a próxima linha do resultado;
    - Este é o único caminho que podemos percorrer com o cursor;
  - Usamos a cláusula INTO para guardar o resultado da operação em uma ou mais variáveis;
    - Da mesma forma que a usamos em uma seleção;
    - A variável ou lista de variáveis devem ser compatíveis com a linha do resultado da consulta;
  - Exemplo:

```
OPEN c1 FOR SELECT horas_trabalhadas
                   FROM empregado e, trabalha_projeto tp
                   WHERE e.id=tp.id_emp;
FETCH c1 INTO horas;
```

Stored Procedures

- O tipo **Cursor**:
  - Ao terminamos as tarefas relacionadas ao cursor, devemos fechar o mesmo para liberar o espaço de memória que ele utiliza;
  - Ou para liberar a variável para que ela possa ser aberta novamente;
  - Fazemos isso através do operador CLOSE, que tem, para os dois tipos de cursor, a seguinte sintaxe:

```
CLOSE nomeDoCursor;
```

Triggers

- Um gatilho é um tipo especial de procedimento armazenado que executa quando um determinado evento ocorre no banco de dados;
- É composto pela seqüência: Evento -> Condição -> Ação
  - Eventos:
    - São as situações onde o gatilho pode ser disparado;
    - Geralmente são atualizações feitas em tabelas ou visões: INSERT, DELETE ou UPDATE;
  - Condição:
    - É a condição que deve ser satisfeita no momento do evento para que o gatilho seja disparado;
    - Não sendo obrigatório;
  - Ações:
    - São as ações que devem ser executadas pelo SGBD caso o evento aconteça e a condição seja verdadeira;
    - Geralmente, é uma sequência de comandos SQL;
    - Operações em tabelas;
    - Chamadas a procedimentos armazenados;

Triggers

- Em PL/SQL, quando definimos um gatilho, as ações que devem ser disparadas por ele devem ser especificadas em uma função que:
  - Não tenha nenhum parâmetro de entrada;
  - Tenha um 'trigger' como tipo de retorno;

Triggers

- Sempre que a função que representa um gatilho é chamada, ela recebe algumas informações:
- As mais importantes são:
  - NEW e OLD:
    - Variáveis que representam os valores atuais e antigos, respectivamente, de uma linha que causou o disparo do gatilho;
  - TG_NAME:
    - Variável que armazena o nome do gatilho que foi disparado;
  - TG_WHEN:
    - Indica quando o gatilho é disparado;
      - Pode receber os valores AFTER ou BEFORE;
  - TG_LEVEL:
    - Indica o nível do gatilho;
      - Pode receber os valores ROW e STATEMENT;

Triggers

- Sempre que a função que representa um gatilho é chamada, ela recebe algumas informações:
- As mais importantes são:
  - TG_OP:
    - Indica a operação que causou o disparo do gatilho;
      - Pode receber os valores INSERT, UPDATE e DELETE;
  - TG_RELNAME:
    - Indica o nome da tabela que causou o disparo do gatilho;

Triggers

- Uma função que representa um gatilho pode retornar uma linha ou um valor nulo;
- Quando um gatilho é definido a nível de uma tupla, acontece o seguinte:
  - Se ele for disparado antes de uma operação, o valor retornado representa a tupla que vai concluir a operação;
  - Caso contrário, o tipo de retorno é ignorado;
- Gatilhos que não são a nível de tupla também têm seu tipo de retorno ignorado;

- Exemplo: Criar a função “aloca()” de gatilho para assegurar que cada novo empregado incluso na empresa seja alocado para trabalhar no departamento 1

Triggers

- Uma vez criada a função que contém a ação de um gatilho, criamos o gatilho propriamente dito através do comando CREATE TRIGGER;

- Este comando tem a seguinte sintaxe:

```
CREATE TRIGGER nomeGatilho {BEFORE | AFTER}
{INSERT | UPDATE | DELETE} ON nomeDaTabela
FOR EACH {ROW | STATEMENT}
EXECUTE PROCEDURE funcaoDoGatilho;
```

- Exemplo de criação de um gatilho para a nossa função:

```
CREATE TRIGGER aloca_projeto AFTER INSERT
ON empregado
FOR EACH ROW
EXECUTE PROCEDURE aloca();
```

Triggers

- Exemplo: Criar uma função de gatilho para assegurar que para todas as alterações realizadas na tabela usuario, serão registrados um código, a data, o usuário e o tipo de alteração realizada
  - Primeiramente criaremos uma nova tabela que armazenará o log de alterações:

```
CREATE TABLE log_alteracoes(
        id SERIAL,
          data DATE,
          autor VARCHAR(20),
          alteracao VARCHAR(6),
          PRIMARY KEY (id)
);
```

Triggers

- Exemplo 9: Criar uma função de gatilho para assegurar que para todas as alterações realizadas na tabela usuario, serão registrados um código, a data, o usuário e o tipo de alteração realizada
  - Agora, criaremos a procedimento armazenado que será utilizado pelo gatilho:

```
CREATE OR REPLACE FUNCTION gera_log()
RETURNS TRIGGER
AS'
    BEGIN
    INSERT INTO log_alteracoes (data, autor, alteracao)
                        VALUES ( now(), user, TG_OP);
    RETURN NEW;
    END '
LANGUAGE plpgsql;
```

Triggers

- Exemplo 9: Criar uma função de gatilho para assegurar que para todas as alterações realizadas na tabela usuario, serão registrados um código, a data, o usuário e o tipo de alteração realizada
  - Por fim, criamos o nosso gatilho que será disparado quando houver algum evento (Atualização, Remoção, Inserção) na tabela trabalha_projeto:

```
CREATE TRIGGER trigger_gera_log
BEFORE INSERT OR UPDATE OR DELETE
ON usuario
FOR EACH ROW
EXECUTE PROCEDURE gera_log()
```

Triggers

- Exemplo de remoção de um gatilho do banco de dados:
  - DROP TRIGGER nome_gatilho ON tabela;