Banco de Dados

Material de Apoio

Parte XI – Views e Stored Procedure

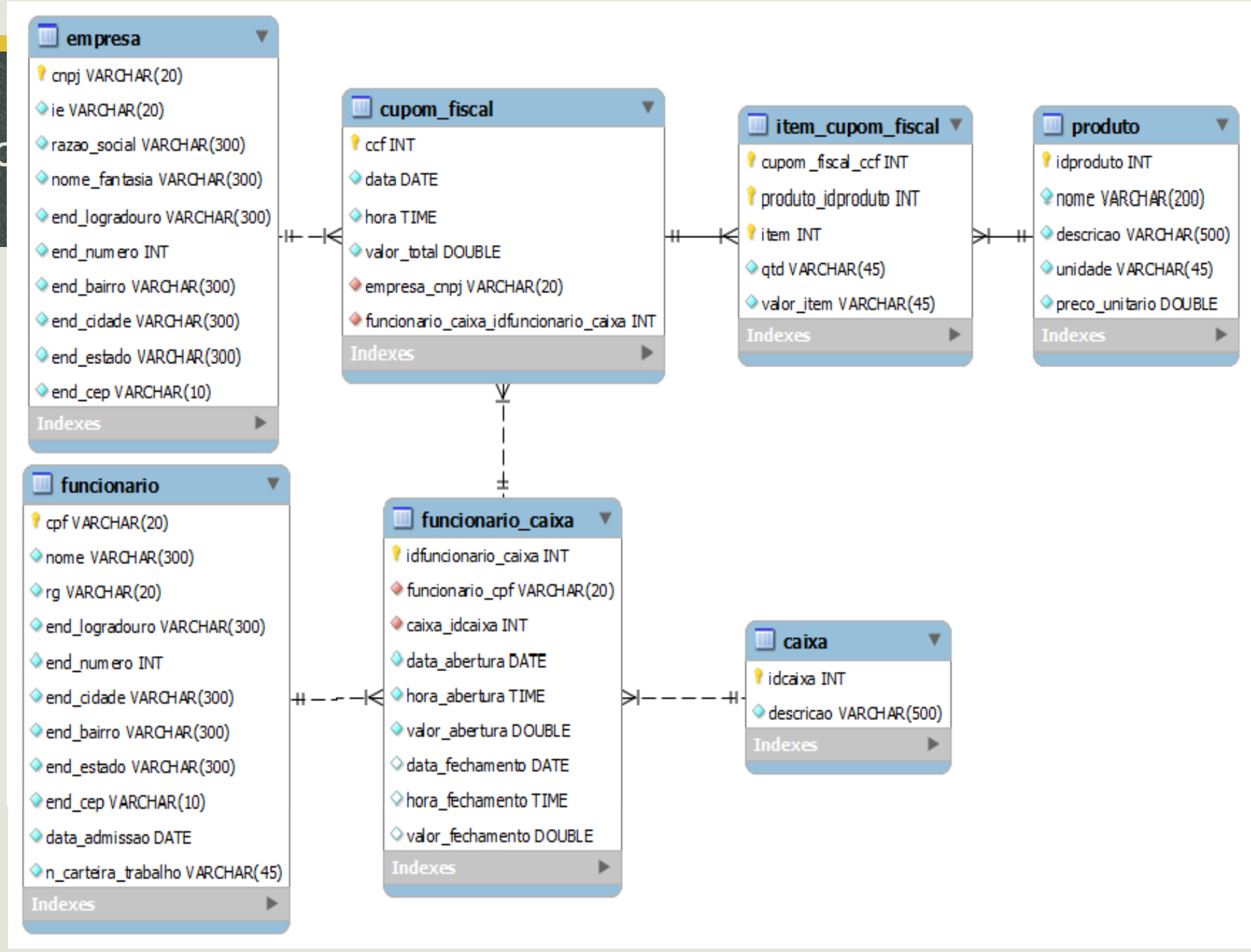
Prof. José Osvano da Silva, PMP, PSM I
joseosvano@unipac.br

1º sem / 2022

Sumário

- Consultas avançadas SQL
- Views em SQL
- Stored Procedure em SQL
- Exercício

Banco de Dado



Operações com Conjuntos

- **union:** seleciona os cupons fiscais dos meses setembro e novembro

```
select * from cupom_fiscal where data between ‘2021-11-01’
and ‘2021-11-30’
union
select * from cupom_fiscal where data between ‘2021-09-01’
and ‘2021-09-30’
```

- **Nota**
  - A união pode ser feito com tabelas e campos diferentes desde que a quantidade e os tipos sejam iguais entre as tabelas da união

Subconsultas aninhadas

- A SQL oferece um mecanismo para aninhar subconsultas
  - **in:** testa se um elemento está presente em algum conjunto
  - **not in:** testa a ausência de um elemento em algum conjunto

Subconsultas aninhadas

Exemplo

- SELECT DISTINCT nome FROM funcionario WHERE nome **IN** (SELECT f.nome FROM funcionario_caixa AS fc, funcionario AS f where f.cpf=fc.funcionario_cpf AND fc.data_fechamento IS NULL);
- SELECT DISTINCT nome FROM funcionario WHERE nome **NOT IN** (SELECT f.nome FROM funcionario_caixa AS fc, funcionario AS f WHERE f.cpf=fc.funcionario_cpf AND fc.data_fechamento IS NULL);

Comparação de Conjuntos

- > some: seleciona as tuplas que são maior que alguma das seleções do argumento some

- SELECT * FROM cupom_fiscal WHERE valor_total > SOME (200, 300, 400);

- O mesmo se aplica as declarações >= some, < some, <= some, <> some

Equivalente:

- SELECT * FROM cupom_fiscal WHERE valor_total > 200 or valor_total > 300 or valor_total > 400;

Comparação de Conjuntos

- > all: seleciona as tuplas que são maior que todas as seleções do argumento all

- SELECT * FROM cupom_fiscal WHERE valor_total > ALL (200, 300, 400);
- O mesmo se aplica as declarações >= all, < all, <= all, <> all

Equivalente:

- SELECT * FROM cupom_fiscal WHERE valor_total > 200 and valor_total > 300 and valor_total > 400;

Teste de relações vazias

- exists: retorna verdadeiro se há pelo menos uma ocorrência na seleção
  - SELECT * FROM cupom_fiscal AS c WHERE EXISTS (SELECT * FROM item_cupom_fiscal AS icf WHERE icf.produto_idproduto = 1 AND c.ccf=icf.cupom_fiscal_ccf);

- not exists retorna verdadeiro se não há ocorrências na seleção
  - SELECT * FROM cupom_fiscal AS c WHERE NOT EXISTS (SELECT * FROM item_cupom_fiscal AS icf WHERE icf.produto_idproduto = 1 AND c.ccf=icf.cupom_fiscal_ccf);

Relações derivadas

- Podemos expressar uma consulta na cláusula from
  - SELECT * FROM (SELECT * FROM cupom_fiscal WHERE valor_total >= 100) AS cfm100 WHERE cfm100.data = ‘2021-11-11';

Views

- Definição
  - Um meio de fornecer ao usuário um modelo de dados personalizado
  - Uma **visão** pode esconder dados que um determinado usuário não precisa ver

Views

- Vantagens:
  - Restringir “quais colunas” de uma tabela podem ser acessadas (leitura/modificação)
  - Permite simplificar consultas complexas para os usuários finais
  - Podem conter valores calculados ou valores de resumo
  - Permite uma maior segurança as informações por permitir que apenas uma parte da informação seja exposta ao usuário

Views

- Desvantagens:
  - Performance: As consultas baseadas em visões podem perder em velocidade, principalmente se a visão for baseada em outras visões
  - Dependência entre tabelas: Se as tabelas da visão forem alteradas, a visão, muito provavelmente, deverá ser alterada também

Views

- Como definimos uma visão?
  - Definição de um **SELECT** que faz uma consulta sobre as tabelas
  - A visão aparece no esquema de dados "como se fosse uma tabela"

Views

- **Sintaxe de criação**
  - **CREATE VIEW** <nome da visao> **AS (<instrução** SELECT> **)**

Views

- Exemplo
  - CREATE VIEW cupons_novembro AS SELECT * FROM cupom_fiscal WHERE data BETWEEN '2018-11-01' AND '2018-11-30';

Views

- Visualizar uma visão
  - SHOW CREATE VIEW [database_name].[view_name];

- Exemplo
  - SHOW CREATE VIEW cupons_novembro;

Views

- Uma vez que uma visão é definida, podemos modificá-la com o comando alter view.

```
ALTER
  [ALGORITHM ={MERGE | TEMPTABLE |   UNDEFINED}]

   VIEW [database_name].  [view_name]  AS
[SELECT statement]
```

Views

- DROP VIEW [IF EXISTS]
  - [database_name].[view_name]

Stored Procedure

- Uma stored procedure é um segmento da SQL declarativa para armazenar funcionalidades que podem ser chamados através de triggers, outros stored procedures ou aplicações escritas em Java, C#, PHP, etc.

Vantagens

- Aumento da performance da aplicação
  - Uma vez criada, a stored procedure é compilada e armazenada no banco de dados
- Reduz o tráfego de dados entre a aplicação e o banco de dados
  - A aplicação só precisa chamar a stored pocedure
- Stored procedures são reusáveis por diversas linguagens de programação
  - Não é preciso reescrever a mesma função em cada aplicação diferente, ela já está implementada no banco
- Podemos administrar as permissão de cada aplicação a stored procedure

Desvantagens

- Se há um número muito grande stored procedures, o tamanho de memória utilizada para cada conexão aumenta substancialmente
- Além disso, se você utilizar um grande número de operações lógicas dentro do stored procedure, o uso da cpu também vai aumentar, pois o servidor de banco de dados não foi projetado para trabalhar com operações lógicas
- Difícil debugar stored procedures
- Não é fácil desenvolver e manter stored procedures e alinhá-los com as necessidades das aplicações

Stored Procedure

- Declarando uma stored procedure

```
CREATE PROCEDURE pegarTodosProdutos()
BEGIN
    SELECT * FROM produto;
END
```

Stored Procedure

- Chamando uma stored procedure
- EXEC STORED_PROCEDURE_NAME();
- EXEC pegarTodosProdutos();

Stored Procedure

- Declarando variáveis
  - DECLARE variable_name datatype(size) DEFAULT default_value;
  - Exemplos
    - DECLARE @total_sale INT = 0;
    - DECLARE @x, @y INT = 0;
- Atribuindo valores
  - Exemplos
    - DECLARE total_count INT = 0
    - SET total_count = 10;

ESCOPO DE VARIÁVEL

- A variável tem escopo da stored procedure onde ela foi declarada
- Se a variável for precedida do caracter @, ela tem escopo de sessão e estará fora do escopo quando a sessão terminar

PARÂMETROS EM STORED PROCEDURES

- A maioria das stored procedures que definimos necessitam de parâmetros
- Os parâmetros podem ser de entrada (IN), saída (OUT) ou entrada e saída (INOUT)
  - IN
    - Quem chama passa argumentos para a stored procedure
    - Os parâmetros são passados por valor
  - OUT:
    - O valor da variável pode ser alterado dentro do stored procedure. Este valor é repassado para a variável de saída (OUT) quando ostored procedure termina
    - O valor inicial da variável OUT não pode ser acessada
  - INOUT:
    - É uma junção das características do IN e do OUT

PARÂMETROS EM STORED PROCEDURES

- Exemplo com IN

```
CREATE PROCEDURE pegarCuponsporData( IN  pdata date)
BEGIN
  SELECT * FROM cupom_fiscal WHERE data =  pdata;
END;

EXEC pegarCuponsporData(‘2021-11-11');
```

PARÂMETROS EM STORED PROCEDURES

- Exemplo com OUT

```
CREATE PROCEDURE qtdCuponsporData(IN @pdata DATE,  OUT pqtd
INT)
BEGIN
  select count(valor_total) into pqtd from cupom_fiscal  where
  data = @pdata;

END;


EXEC qtdCuponsporData('2021-11-11', @qtd1);
select @qtd1;
```

PARÂMETROS EM STORED PROCEDURES

- Exemplo INOUT

```
CREATE PROCEDURE set_counter(INOUT count INT(4), IN  inc INT(4))
BEGIN
    SET count = count + inc;
    END;

SET @counter = 1;
EXEC set_counter(@counter,1);
SELECT @counter;
EXEC set_counter(@counter,5);
SELECT @counter;
```

Declaração IF

- Sintaxe da declaração IF

```
IF if_expression THEN
    commands
[ELSE IF elseif_expression THEN commands]
[ELSE commands]
```

Declaração IF

- Exemplo

```
CREATE PROCEDURE pegarSituacaoCaixa( IN codCaixa INT, IN codFunc
VARCHAR(20), OUT situacao VARCHAR(50))
BEGIN
    DECLARE ABERTO int;
    DECLARE valor_cf, valor_fc, valor_ac double;
    SELECT data_fechamento IS NULL INTO aberto FROM funcionario_caixa WHERE
caixa_idcaixa=codCaixa and funcionario_cpf like codFunc;

    SELECT SUM(valor_total) INTO valor_cf from funcionario_caixa AS fc, cupom_fiscal AS cf where
caixa_idcaixa=codCaixa AND funcionario_cpf like codFunc AND fc.idfuncionario_caixa =
cf.funcionario_caixa_idfuncionario_caixa;

    SELECT valor_abertura INTO valor_ac from funcionario_caixa WHERE caixa_idcaixa=codCaixa
AND funcionario_cpf like codFunc;
```

Declaração IF

```
SELECT valor_fechamento INTO valor_fc from funcionario_caixa where
caixa_idcaixa=codCaixa AND funcionario_cpf LIKE codFunc;


IF(aberto = 1) THEN
    SET situacao = 'ABERTO';
ELSEIF ((valor_fc - valor_ac) <> valor_cf) THEN

SET situacao = 'FECHADO_INCONSISTENTE'; ELSE

SET situacao = 'FECHADO'; END IF;

END;
```

Declaração IF

```
EXEC pegarSituacaoCaixa(1, '200', @sit);  select @sit;
```

Declaração CASE

```
CASE
  WHEN condition_1 THEN commands   WHEN condition_2
  THEN commands
  ...
  ELSE commands
END CASE;
```

Declaração CASE

- Exemplo

```
DELIMITER $$
CREATE PROCEDURE getMensagemSituacaoCaixa( in situ varchar(40), out  msg varchar(100))
 BEGIN
        CASE situ
                WHEN 'ABERTO' THEN
                  SET msg = 'O caixa ainda se encontra ABERTO';
                WHEN 'FECHADO_INCONSISTENTE' THEN
                  SET msg = 'O caixa se encontra FECHADO COM INCONSITÊNCIAS';
                WHEN 'FECHADO' THEN
                  SET msg = 'O caixa se encontra FECHADO';
                ELSE
                  SET msg = 'OPÇÃO INVÁLIDA';
        END CASE;

 END $$ DELIMITER ;
```

Exercício

Basei-se nas Relações Abaixo:

ambulatorio (numeroa, andar, capacidade)

cidade (codcidade, descricao, uf)

doenca (coddoenca, descricao)

especialidade (codesp, nome)

medico (crm, nome, idade, #codcidade, #codesp, #numeroa)

paciente (rg, nome, idade, #codcidade)

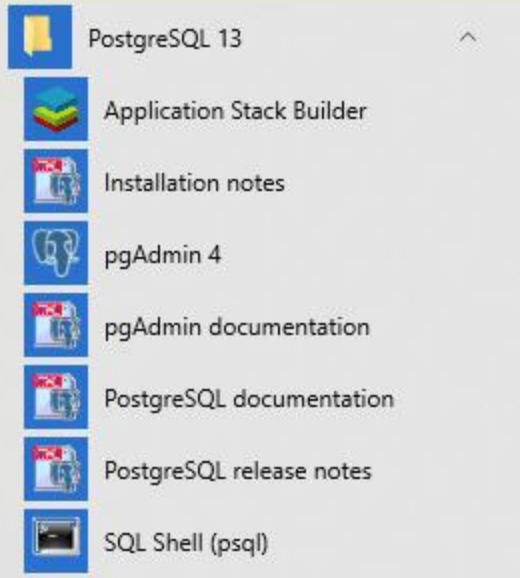
consulta (crm, rg, data, hora, #coddoenca)

funcionario (rg, nome, idade, #codcidade, salario)

Obs.:

Crie um SGBD no PostgreSQL chamado CONSULTAS;

Utilizem os scripts enviados para criar e popular as tabelas;



Dúvidas

José Osvano da Silva
joseosvano@unipac.br