

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
ESCOLA AGRÍCOLA DE JUNDIAÍ
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CARLA FERNANDES CURVELO
CARLAFCF@GMAIL.COM

TAD0010 – BANCO DE DADOS II

AULA 4

VISÕES, GATILHOS E PROCED. ARMAZENADOS

VISÕES

VISÕES (VIEWS)

- ▶ É uma única tabela que é derivada de outras tabelas
 - ▶ Podem ser derivadas de tabelas de base ou de outras *VIEWS*
- ▶ É considerada uma **tabela virtual**, não necessariamente existe em forma física
 - ▶ Consultas podem ser feitas normalmente
 - ▶ Atualizações nem sempre podem ser feitas
- ▶ É uma forma de especificar uma tabela que utilizamos com frequência, mas que não existe fisicamente

VISÕES (VIEWS)

- ▶ A *view* não é realizada ou materializada no momento de sua definição, mas sim quando é especificada uma consulta na *view*
- ▶ É responsabilidade do SGBD, e não do usuário, manter a *view* atualizada
- ▶ Uma *view* pode ser descartada utilizando o comando **DROP VIEW** NOME_DA_VIEW
- ▶ Posso fazer mudanças na *view* (DELETE, UPDATE)?

VISÕES (VIEWS)

PERSONAGEM (Pnome, Unome, Codigo, Datnasc, Residencia, Sexo, Valor, Codigo_chefe, Cnr)

EPISODIO (Episodio_nome, Episodio_numero, Episodio_local, Cnum)

TRABALHA_EM (Pcodigo, Enr)

- ▶ Podemos frequentemente consultar dados sobre o episódio em que um determinado personagem trabalha.
- ▶ Ao invés de fazer repetidamente uma junção entre as tabelas **PERSONAGEM**, **EPISODIO**, **TRABALHA_EM** (tabelas de definição), criar uma *view* com essa junção
- ▶ Depois, realizamos consultas sobre a *view* (leituras sobre uma única tabela, e não sobre várias tabelas)

ESPECIFICAÇÃO DE VIEWS EM SQL

PERSONAGEM (Pnome, Unome, Codigo, Datnasc, Residencia, Sexo, Valor, Codigo_chefe, Cnr)

EPISODIO (Episodio_nome, Episodio_numero, Episodio_local, Cnum)

TRABALHA_EM (Pcodigo, Enr)

CASA (Cnome, Cnumero)

CREATE VIEW TRABALHA_EM1

AS SELECT Pnome, Unome, Episodio_nome, Valor

FROM PERSONAGEM P, EPISODIO E, TRABALHA_EM T

WHERE P.Codigo = T.Pcodigo **AND** E.Episodio_numero = T.Enr;

CREATE VIEW CASA_INFO (Casa_nome, Qnt_pers, Total_valor)

AS SELECT Cnome, **COUNT**(*), **SUM**(Valor)

FROM PERSONAGEM P, CASA C

WHERE P.Cnr = C.Cnumero

GROUP BY Cnumero;

ESPECIFICAÇÃO DE VIEWS EM SQL

PERSONAGEM (Pnome, Unome, Codigo, Datnasc, Residencia, Sexo, Valor, Codigo_chefe, Cnr)

EPISODIO (Episodio_nome, Episodio_numero, Episodio_local, Cnum)

TRABALHA_EM (Pcodigo, Enr)

CASA (Cnome, Cnumero)

CREATE VIEW TRABALHA_EM1

AS SELECT Pnome, Unome, Episodio_nome, Valor

FROM PERSONAGEM P, EPISODIO E, TRABALHA_EM T

WHERE P.Codigo = T.Pcodigo **AND** E.Episodio_numero = T.Enr;

Especificar o primeiro e último nome (Pnome, Unome) de todos os personagens que trabalharam em um episódio com o nome (Episodio_nome) 'Red Wedding'.

SELECT Pnome, Unome

FROM TRABALHA_EM1

WHERE Episodio_nome = 'Red Wedding';

VISÕES (VIEWS)

- ▶ Como o SGBD implementa as *views* eficientemente?
- ▶ **Estratégia 1:** modificação de consulta
 - ▶ Transforma a consulta em *views* em uma consulta em tabelas base
 - ▶ Desvantagem: ineficiente para consultas complexas (demoradas de executar), principalmente se várias consultas às mesmas tabelas são feitas em um curto período

VISÕES (VIEWS)

- ▶ Como o SGBD implementa as *views* eficientemente?
- ▶ **Estratégia 2:** materialização de *view*
 - ▶ Cria temporariamente uma tabela de *views* quando a *view* for consultada pela primeira vez, e mantém para consultas seguintes
 - ▶ **Como atualizar essa tabela quando as tabelas da base forem alteradas?**
 - ▶ **Atualização incremental:** altera as tabelas de *views* materializadas quando as tabelas da base forem alteradas
 - ▶ Depois de um tempo a tabela de *view* materializada é removida automaticamente

SQL – EXEMPLOS

CASA (Cnome, Cnumero)

PERSONAGEM (Pnome, Unome, Codigo, Datnasc, Residencia, Sexo, Valor, Codigo_chefe, Cnr)

RESPONSAVEL (Codigo_responsavel, Casa_numero, Data_inicio)

EPISODIO (Episodio_nome, Episodio_numero, Episodio_local, Cnum)

TRABALHA_EM (Pcodigo, Enr)

DEPENDENTE (Pcodigo, Nome_dependente, Sexo, Datnasc, Parentesco)

CRIAÇÃO DE VIEWS

CASA (Cnome, Cnumero)

PERSONAGEM (Pnome, Unome, Codigo, Datnasc, Residencia, Sexo, Valor, Codigo_chefe, Cnr)

RESPONSAVEL (Codigo_responsavel, Casa_numero, Data_inicio)

Crie uma *view* que tem o nome da casa (Cnome), o nome do responsável da casa (Pnome, Unome) e o seu valor

CRIAÇÃO DE VIEWS

CASA (Cnome, Cnumero)

PERSONAGEM (Pnome, Unome, Codigo, Datnasc, Residencia, Sexo, Valor, Codigo_chefe, Cnr)

RESPONSAVEL (Codigo_responsavel, Casa_numero, Data_inicio)

Crie uma *view* que tem o nome da casa (Cnome), o nome do responsável da casa (Pnome, Unome) e o seu valor

CREATE VIEW RESPONSABIL_CASA

AS SELECT Pnome, Unome, Cnome, Valor

FROM CASA C, PERSONAGEM P, RESPONSABIL R

WHERE C.Cnumero = R.Casa_numero **AND** P.Codigo = R.Codigo_responsavel;

TRIGGER

TRIGGERS

- ▶ Especificar um tipo de ação a ser tomada quando certos eventos ocorrem ou quando certas condições são satisfeitas
- ▶ Exemplo: indicar ao gerente quando as despesas de um funcionário excederem o limite
- ▶ Comando: **CREATE TRIGGER**

```
DELIMITER $$
```

```
CREATE TRIGGER NOME_TRIGGER
```

```
    BEFORE/AFTER INSERT/DELETE/UPDATE ON TABELA
```

```
    FOR EACH ROW
```

```
    BEGIN
```

```
    CORPO;
```

```
END $$
```

```
DELIMITER ;
```

SQL – EXEMPLOS

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

TRIGGER

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

- ▶ **Sal_total** é um atributo derivado
- ▶ Quando ele deve ser atualizado?
 - ▶ Inserção de novos funcionários
 - ▶ Alteração do salário de algum funcionário
 - ▶ Alteração do departamento de algum funcionário
 - ▶ Remoção de algum funcionário

TRIGGER – INSERÇÃO DE NOVOS FUNCIONÁRIOS

DELIMITER \$\$

CREATE TRIGGER INSERIR_FUNCIONARIO

AFTER INSERT ON FUNCIONARIO

FOR EACH ROW

BEGIN

IF (**NEW**.ID_Depto **IS NOT NULL**) **THEN**

UPDATE DEPARTAMENTO

SET Sal_total = Sal_total+**NEW**.Salario

WHERE ID=NEW.ID_Depto;

END IF;

END \$\$

DELIMITER ;

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

TRIGGER – ALTERAÇÃO DO SALÁRIO OU DEPARTAMENTO DE ALGUM FUNCIONÁRIO

DELIMITER \$\$

CREATE TRIGGER ALTERAR_FUNCIONARIO

AFTER UPDATE ON FUNCIONARIO

FOR EACH ROW

BEGIN

IF (**NEW**.ID_Depto = **OLD**.ID_Depto) **THEN**

IF (**NEW**.Salario <> **OLD**.Salario) **THEN**

UPDATE DEPARTAMENTO

SET Sal_total = Sal_total+**NEW**.Salario-**OLD**.Salario

WHERE ID=**NEW**.ID_Depto;

END IF;

ELSE

UPDATE DEPARTAMENTO

SET Sal_total = Sal_total+**NEW**.Salario

WHERE ID=**NEW**.ID_Depto;

UPDATE DEPARTAMENTO

SET Sal_total = Sal_total-**OLD**.Salario

WHERE ID=**OLD**.ID_Depto;

END IF;

END \$\$

DELIMITER ;

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

TRIGGER – REMOÇÃO DE ALGUM FUNCIONÁRIO

DELIMITER \$\$

```
CREATE TRIGGER REMOVER_FUNCIONARIO  
  AFTER DELETE ON FUNCIONARIO  
  FOR EACH ROW  
  BEGIN  
    UPDATE DEPARTAMENTO  
    SET Sal_total = Sal_total-OLD.Salario  
    WHERE ID=OLD.ID_Depto;  
  END $$
```

DELIMITER ;

```
FUNCIONARIO (ID, Nome, Salario, ID_Depto, Observacao)  
DEPARTAMENTO (ID, Nome, Sal_total)  
GERENCIA (ID_Funcionario, ID_Depto)  
SUPERVISOR (ID_Funcionario, ID_Supervisor)
```

EXERCÍCIO

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)
DEPARTAMENTO (**ID**, Nome, Sal_total)
GERENCIA (ID_Funcionario, ID_Depto)
SUPERVISOR (ID_Funcionario, ID_Supervisor)

- ▶ Verificar se um funcionário tem o salário maior do que o salário do seu supervisor. Se acontecer, indicar isso nas observações
- ▶ Que opções temos?
 - ▶ Associar um supervisor a um novo funcionário
 - ▶ Alterar o supervisor de um funcionário
 - ▶ Remoção de uma relação de supervisão
 - ▶ Alterar o salário de um funcionário

TRIGGER – INSERÇÃO DE NOVOS SUPERVISORES

DELIMITER \$\$

CREATE TRIGGER INSERIR_SUPERVISOR

AFTER INSERT ON SUPERVISOR

FOR EACH ROW

BEGIN

DECLARE Salario_Funcionario DECIMAL(10,2);

DECLARE Salario_Supervisor DECIMAL(10,2);

SET @Salario_Funcionario = (**SELECT** Salario **FROM** FUNCIONARIO **WHERE** ID=**NEW**.ID_Funcionario);

SET @Salario_Supervisor = (**SELECT** Salario **FROM** FUNCIONARIO **WHERE** ID=**NEW**.ID_Supervisor);

IF (@Salario_Funcionario>@Salario_Supervisor) **THEN**

UPDATE FUNCIONARIO

SET Observacao='Salario maior do que o do supervisor'

WHERE ID=**NEW**.ID_Funcionario;

END IF;

END \$\$

DELIMITER ;

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

TRIGGER – ALTERAÇÃO O SUPERVISOR DE ALGUM FUNCIONÁRIO

DELIMITER \$\$

CREATE TRIGGER ALTERAR_SUPERVISOR

AFTER UPDATE ON SUPERVISOR

FOR EACH ROW

BEGIN

DECLARE Salario_Funcionario DECIMAL(10,2);

DECLARE Salario_Supervisor DECIMAL(10,2);

SET @Salario_Funcionario = (**SELECT** Salario **FROM** FUNCIONARIO **WHERE** ID=**NEW**.ID_Funcionario);

SET @Salario_Supervisor = (**SELECT** Salario **FROM** FUNCIONARIO **WHERE** ID=**NEW**.ID_Supervisor);

IF (**NEW**.ID_Funcionario <> **OLD**.ID_Funcionario) **THEN**

UPDATE FUNCIONARIO

SET Observacao=NULL

WHERE ID=**OLD**.ID_Funcionario;

IF (@Salario_Funcionario>@Salario_Supervisor) **THEN**

UPDATE FUNCIONARIO

SET Observacao='Salario maior do que o do supervisor'

WHERE ID=**NEW**.ID_Funcionario;

END IF;

ELSEIF (**NEW**.ID_Supervisor <> **OLD**.ID_Supervisor) **THEN**

IF (@Salario_Funcionario>@Salario_Supervisor) **THEN**

UPDATE FUNCIONARIO

SET Observacao='Salario maior do que o do supervisor'

WHERE ID=**NEW**.ID_Funcionario;

ELSE

UPDATE FUNCIONARIO

SET Observacao=NULL

WHERE ID=**NEW**.ID_Funcionario;

END IF;

END IF;

END \$\$

DELIMITER ;

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

TRIGGER – REMOVER A RELAÇÃO DE SUPERVISÃO

DELIMITER \$\$

```
CREATE TRIGGER REMOVER_SUPERVISAO  
  AFTER DELETE ON SUPERVISOR  
  FOR EACH ROW  
  BEGIN  
    UPDATE FUNCIONARIO  
    SET Observacao=NULL  
    WHERE ID=OLD.ID_Funcionario;  
  END $$
```

DELIMITER ;

```
FUNCIONARIO (ID, Nome, Salario, ID_Depto, Observacao)  
DEPARTAMENTO (ID, Nome, Sal_total)  
GERENCIA (ID_Funcionario, ID_Depto)  
SUPERVISOR (ID_Funcionario, ID_Supervisor)
```

TRIGGER – ALTERAR O SALÁRIO DE UM FUNCIONÁRIO

DELIMITER \$\$

CREATE TRIGGER ALTERAR_SALARIO

BEFORE UPDATE ON FUNCIONARIO

FOR EACH ROW

BEGIN

DECLARE Salario_como_Funcionario DECIMAL(10,2);

DECLARE Salario_do_Supervisor DECIMAL(10,2);

IF (NEW.Salario <> OLD.Salario) THEN

SET @Salario_como_Funcionario = **NEW.Salario**;

SET @Salario_do_Supervisor = (**SELECT** F.Salario **FROM** SUPERVISOR S, FUNCIONARIO F **WHERE**
S.ID_Supervisor=F.ID **AND** S.ID_Funcionario=**NEW.ID**);

IF(@Salario_do_Supervisor **IS NOT NULL**) **THEN**

IF (@Salario_como_Funcionario>@Salario_do_Supervisor) **THEN**

SET NEW.Observacao='Salario maior do que o do supervisor';

ELSE

SET NEW.Observacao=NULL;

END IF;

END IF;

END IF;

END \$\$

DELIMITER ;

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

EXERCÍCIO

FUNCIONARIO (**ID**, Nome, Salario, ID_Depto, Observacao)

DEPARTAMENTO (**ID**, Nome, Sal_total)

GERENCIA (ID_Funcionario, ID_Depto)

SUPERVISOR (ID_Funcionario, ID_Supervisor)

- ▶ Verificar se um funcionário é gerente de um departamento. Se for, adicionar nas observações a informação "Gerente de departamento".
- ▶ Caso já tenha alguma observação, adicionar essa observação ao texto já existente.
- ▶ Que opções temos?
 - ▶ Definir um novo gerente a um departamento
 - ▶ Alterar a gerência de um departamento
 - ▶ Remover a gerência de um departamento

VARCHAR

CONCAT(texto1, texto2)

REPLACE(entrada, procura, substituir)

**PROCEDIMIENTOS
ARMAZENADOS**

PROCEDIMENTOS ARMAZENADOS

- ▶ É comum em uma aplicação de BD executar rotinas complexas utilizando várias instruções SQL em sequência
- ▶ Pode ter muito consumo de recursos pela aplicação
 - ▶ Em aplicação WEB isso é mais visível ainda
- ▶ Como atenuar parte deste consumo de recursos direto da aplicação?
 - ▶ Transferir parte do processamento direto para o banco de dados
 - ▶ Máquinas servidoras geralmente têm configurações de hardware mais robustas

PROCEDIMENTOS ARMAZENADOS

- ▶ **Stored Procedures** (ou Procedimentos Armazenados)
 - ▶ Stored procedures são rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser invocadas
 - ▶ Um procedimento desses pode executar uma série de instruções, receber parâmetros e retornar valores

PROCEDIMENTOS ARMazenados



CLIENTE

ATUALIZAR STATUS PEDIDO →

ATUALIZAR STATUS ITENS →

LANÇAR NO CAIXA →



SERVIDOR / BD



CLIENTE

CONFIRMAR PEDIDO →



SERVIDOR / BD

(
ATUALIZAR STATUS PEDIDO
ATUALIZAR STATUS ITENS
LANÇAR NO CAIXA
)

PROCEDIMENTOS ARMAZENADOS

▶ Pontos positivos:

- ▶ Simplificação da execução de instruções **SQL** pela aplicação
- ▶ Transferência de parte da responsabilidade de processamento para o servidor
- ▶ Facilidade na manutenção, reduzindo a quantidade de alterações na aplicação

PROCEDIMENTOS ARMazenados

► Comando: **CREATE PROCEDURE**

DELIMITER \$\$

CREATE PROCEDURE NOME_PROCEDURE

BEGIN

CORPO;

END \$\$

DELIMITER ;

PROCEDIMENTOS ARMazenados

DELIMITER \$\$

CREATE PROCEDURE ATUALIZAR_OBS_FUNCIONARIO(**IN** ID_Funcionario INT, **IN** OPERACAO BIT)

BEGIN

IF (OPERACAO=0) **THEN**

UPDATE FUNCIONARIO

SET Observacao='Salario maior do que o do supervisor'

WHERE ID=ID_Funcionario;

ELSE

UPDATE FUNCIONARIO

SET Observacao=NULL

WHERE ID=ID_Funcionario;

END IF;

END \$\$

DELIMITER ;

CALL ATUALIZAR_OBS_FUNCIONARIO(1, 0);

CALL ATUALIZAR_OBS_FUNCIONARIO(2, 1);

PROCEDIMENTOS ARMazenados

DELIMITER \$\$

CREATE PROCEDURE LER_SALARIO_FUNCIONARIO(**IN** ID_Funcionario INT, **OUT** Salario_Funcionario DECIMAL(10,2))

BEGIN

 SELECT Salario INTO Salario_Funcionario
 FROM FUNCIONARIO
 WHERE ID= ID_Funcionario;

END \$\$

DELIMITER ;

CALL LER_SALARIO_FUNCIONARIO(4, @Salario_Funcionario);

SELECT @Salario_Funcionario;

PROCEDIMENTOS ARMazenados

DELIMITER \$\$

CREATE PROCEDURE ATUALIZAR_SALARIO_FUNCIONARIO(**IN** ID_Funcionario INT, **INOUT** Salario_Funcionario DECIMAL(10,2))

BEGIN

DECLARE Salario_atual DECIMAL(10,2);

SET @Salario_atual = (**SELECT** Salario **FROM** FUNCIONARIO **WHERE** ID = ID_Funcionario);

SET Salario_Funcionario=@Salario_atual + Salario_Funcionario;

END \$\$

DELIMITER ;

SET @Salario_Funcionario=200;

CALL ATUALIZAR_SALARIO_FUNCIONARIO(1, @Salario_Funcionario);

SELECT @Salario_Funcionario;

BIBLIOGRAFIA

- ▶ ELMASRI, Ramez; NAVATHE, Sham. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson, 2011. 788 p. [cap. 5 e 26]