Visões, triggers e procedimentos armazenados

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

FUNDAMENTOS DE BANCO DE DADOS

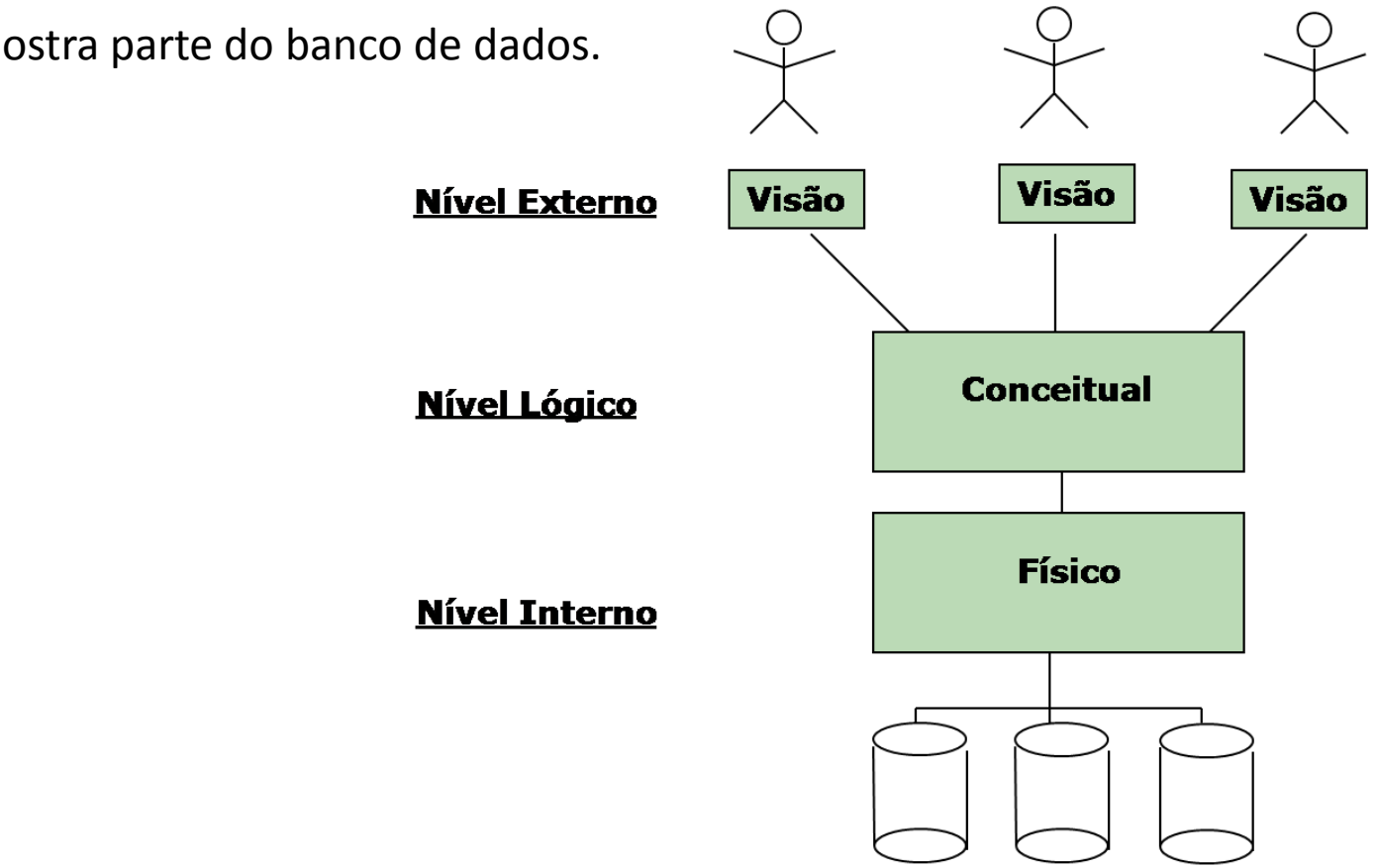
PROFA. LÍVIA ALMADA

Visões

- Acesso a um banco de dados
  - Requer conhecimento do esquema
- Indesejável
  - Para usuários inexperientes
  - Desenvolvedores de aplicativos que acessam o BD
- Por questões de segurança e privacidade
  - Grupos de usuários devem ter acesso a dados de interesse
- O acesso a todo o banco de dados é perigoso

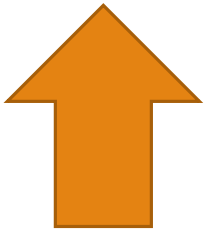
Visões

Cada uma mostra parte do banco de dados.



Visões

FUNC_DEP(funcnome, depnome)



FUNCIONARIO(cod, pnome, unome, coddep)

DEP(cod, dnome)

Consulta sobre visões

FUNC_DEP(funcnome, depnome)

SELECT * FROM FUNC_DEP WHERE depnome like ‘%informatica’;

Conceito de visão

- Conceito de uma visão em SQL:
  - Tabela única geralmente derivada de outras tabelas.
  - Considerada como uma tabela virtual.
- Usos
  - Simplifica a especificação de certas consultas.
  - Mecanismo de segurança e autorização.
- Comando CREATE VIEW
  - Define um nome de tabela, uma lista de nomes de atributos e uma consulta para especificar o conteúdo da visão.
  - A visão está sempre atualizada.
  - O comando DROP VIEW elimina uma visão.

Criando um visão

COMANDO CREATE VIEW

```
Create or  replace   view func_dep as(
Select pnome || ‘ ‘ || ‘ ‘ || unome as funcnome,
                     dnome as  depart
                     from funcionário f,  departamento d
                     where      dnumero = dnr
);
```

```
Create or  replace   view func_dep (funcnome, dnome) as(
Select pnome || ‘ ‘ || ‘ ‘ || unome , dnome
                     from funcionário f,  departamento d
                     where      dnumero = dnr
);
```

SQL – Visões

```
CREATE  VIEW nome_da_visão
[(nome_coluna {, nome_coluna ...})]
AS
        subquery [WITH CHECK OPTION]
```

- WITH CHECK OPTION
  - Especifica que atualizações (INSERT ou UPDATE) na tabela base só serão permitidas se resultam em tuplas visíveis para a visão.

Implementação e atualização

1. Modificação de consulta
   - Modifica a consulta da view por uma consulta nas tabelas base.
   - Desvantagem: Ineficiente para visões definidas por consultas complexas que têm execução demorada.
2. Estratégia de materialização de view
   - Criar fisicamente uma tabela de view temporária quando a view for consultada pela primeira vez.
   - Manter essa tabela na suposição de que outras consultas a view acontecerão em seguida
   - Requer estratégia eficiente para atualizar automaticamente a tabela da view quando as tabelas de base forem atualizadas

Implementação e atualização

- Atualização de visão é complicada e pode ser ambígua.
- Atualização em uma visão definida sobre uma única tabela sem funções agregadas.
  - Pode ser mapeada para uma atualização na tabela base.
- Visões envolvendo junções
  - Frequentemente não é possível o SGBD determinar qual das atualizações é pretendida.
- Cláusula WITH CHECK OPTION
  - Precisa ser acrescentada ao final da definição da visão, se uma visão tiver de ser atualizada.

Atualização

```
UV1:  UPDATE TRABALHA_EM1
      SET      Projnome = 'ProdutoY'
      WHERE    Unome='Silva' AND Pnome='João'
               AND Projnome='ProdutoX';
```

Atualizações podem ser ambíguas...
Atualização a) ou b)?

```
(a): UPDATE TRABALHA_EM
     SET      Pnr =     ( SELECT   Projnumero
                          FROM     PROJETO
                          WHERE    Projnome=
                                   'ProdutoY' )
     WHERE    Fcpf IN   ( SELECT   Cpf
                          FROM     FUNCIONARIO
                          WHERE    Unome='Silva'
                                   AND
                                   Pnome='João' )
              AND
              Pnr =     ( SELECT   Projnumero
                          FROM     PROJETO
                          WHERE    Projnome=
                                   'ProdutoX' );
```

```
(b): UPDATE PROJETO   SET      Projnome =
                               'ProdutoY'
     WHERE      Projnome = 'ProdutoX';
```

Visão virtual

A definição da visão é armazenada

- Dados da visão não são persistentes

- Sempre que referenciada os dados são materializados
  - Custo praticamente igual a cada materialização

- Somente leitura
  - Visões que só permitem acesso de leitura

- Permitem atualização
  - Visões que permitem atualizações nas tabelas base

Visão materializada

- Dados e definição são persistentes
  - Problema de atualização dos dados da visão sempre que há uma atualização nas tabelas base da visão
    - Recalculada
    - Atualizada
      - Com intervenção humana
      - Automática
- Reduz custos de materialização de resultado Visões somente para leitura
- Aplicações
  - Implementação Data Warehouse
  - Integração de fontes de dados heterogêneas

Exemplos

```
create view V1 (nome_departamento, nome_empregado)
    as select d.nome,e.nome
        from Departamento d inner join Empregado e
                on d.id=e.id_dept

create view V2 (nome_empregado, número_de_dependentes)
    as select e.nome, (select count(*) from Dependente
      where                                  matr_resp=e.matr)
            from Empregado e

create view V3 (matrícula, salário)
    as select nome,salário
        from Empregado
         where salário<700 with check option
```

Exemplos

```
select * from V1
select * from V2 where número_de_dependentes>2
select * from V3
```

matrícula	salário
caio	500.0
rebeca	500.0

Exemplos

```
create view V3 (matrícula, salário)
   as select nome,salário
      from Empregado
       where salário<700 with check option
```

```
update v3 set salário=salário+100
   select * from V3
      matrícula                      salário
      ------------------------------ ----------
      caio                            600.0
      rebeca                          600.0
update v3 set salário=salário+150
```

Erro pois tuplas alteradas deixam de fazer parte da visão.

Triggers

- Triggers (gatilhos) são regras que especificam ações que são disparadas automaticamente por certos eventos.
  - Ex. Após um insert, Antes de um update
- Funcionalidades fornecidas pelos bancos de dados ativos estão disponíveis na forma de triggers (gatilhos)
  - Fazem parte da SQL-99 e de padrões mais recentes.

Especificando ações como triggers

Modelo Evento-Condição-Ação (ECA) usado para especificar regras de banco de dados ativo:

- O(s) evento (s) que dispara(m) a regra.
  - Normalmente são operações de atualização do banco de dados.
- A condição que determina se ação da regra deve ser executada.
  - Condição opcional pode ser avaliada, especificada na cláusula WHEN de um trigger.
  - Se nenhuma condição for especificada, a ação será executada quando ocorrer o evento.
  - Se uma condição for especificada, ela é primeiro avaliada e, somente se for avaliada como verdadeira, a ação da regra será executada.
- A ação a ser tomada.
  - Normalmente é uma sequência de comandos SQL
  - Também poderia ser uma transação do banco de dados ou um programa externo que será executado automaticamente.

Sintaxe PostgreSQL

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF }
{ event [ OR ... ] }
    ON table
    [ FROM referenced_table_name ]
    { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE |
INITIALLY DEFERRED } }
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE PROCEDURE function_name ( arguments )
```

event pode ser:

- INSERT
- UPDATE [OF column_name [, ...]]
- DELETE
- TRUNCATE (remove todas as linhas de uma ou mais tabelas)

Sintaxe PostgreSQL

No PostgreSQL é preciso inicialmente criar uma função que retorna uma trigger.

```
create or replace function remove_empr() returns trigger as $$
Begin
delete from departamento where gerente=OLD.cpf;
delete from dependentes  where emp    =OLD.cpf;
delete from trabalha_no  where emp    =OLD.cpf;
return OLD;
end;
$$ language plpgsql;
```

```
create trigger remove_empregado
before delete on empregado
for each row execute procedure remove_empr();
```

Sintaxe PostgreSQL

```
create or replace function insere_depart() returns trigger as $$
begin
if (select count(cpf) from empregado where cpf =NEW.gerente) = 0
then
  RAISE EXCEPTION 'Inserção do departamento com gerente % não pode
ser realizada, pois não existe empregado com esse cpf.',
NEW.gerente;
end if;
return NEW;
end;
$$ language plpgsql;
```

```
create trigger insere_departamento
before insert on departamento
for each row execute procedure insere_depart();
```

Sintaxe PostgreSQL

```
create or replace function atualiza_empr() returns trigger as $$
begin
IF NEW.cpf != OLD.cpf then
  update departamento set gerente=NEW.cpf where gerente=OLD.cpf;
  update dependentes  set emp    =NEW.cpf where emp    =OLD.cpf;
  update trabalha_no  set emp    =NEW.cpf where emp    =OLD.cpf;
  update empregado    set superv =NEW.cpf where superv = OLD.cpf;
end if;
return NEW;
end;
$$ language plpgsql;
```

```
drop trigger if exists atualiza_empregado on empregado;
create trigger atualiza_empregado
after update on empregado
for each row execute procedure atualiza_empr();
```

Especificando ações como triggers

- BEFORE ou AFTER: especifica que a regra será disparada antes ou depois, respectivamente, que ocorrerem os eventos que disparam a regra.
- INSERT, DELETE e UPDATE : eventos básicos para disparar as regras.
  - No caso de UPDATE, podem-se especificar os atributos a serem atualizados.
- ON: determina a relação em que a regra é especificada.
- FOR EACH ROW : a regra será disparada uma vez para cada linha que é afetada pelo evento de disparo.
  - trigger de nível de linha
- FOR EACH STATEMENT: dispara a regra apenas uma vez, mesmo que várias tuplas sejam afetadas pelo evento de disparo
  - trigger em nível de comando que

Especificando ações como triggers

```
CREATE TRIGGER salario_total1
 AFTER UPDATE OF salario ON FUNCIONARIO
 REFERENCING OLD ROW AS O, NEW ROW AS N
 FOR EACH ROW
 WHEN (N.id_dept IS NOT NULL)
 UPDATE DEPARTAMENTO
 SET salario_total = salario_total + N.salario - O.salario
 WHERE id_dept = N.id_dept;
```

Especificando ações como triggers

- REFERENCING: nomeamos variáveis de tupla (apelidos) para nos referirmos à(s) tupla(s) OLD (antes da modificação) e à(s) tupla(s) NEW (após a modificação), respectivamente.
- WHEN é usada para especificar quaisquer condições que precisam ser verificadas após a regra ser disparada, mas antes que a ação seja executada.
  - A ação especificada após a condição pode ser um comando SQL ou uma sequência de comandos SQL delimitados por BEGIN e END.

Referências

- Elsmari, R., Navathe, Shamkant B. “Sistemas de Banco de Dados”. 6ª Edição, Pearson Brasil, 2011. Capítulo 5
- Slides do Prof. Régis Pires