Universidade Federal do Ceará
Campus Crateús

Fundamentos de Banco de Dados

Aula 11 - Integridade de Banco de Dados

Professora Vitória Regina - vitoria@crateus.ufc.br

O que estudaremos?

- Restrições de tuplas.
- Asserções.
- Stored procedures (procedimentos armazenados)
- Gatilhos.

Introdução

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

Restrições de tuplas

- Podemos especificar restrições de tuplas usando o comando CHECK.
- Exemplo:

```
CREATE TABLE Empregado (
        Matricula VARCHAR(10),
        Nome VARCHAR(30) NOT NULL,
        Salario REAL,
      Supervisor VARCHAR(10),
        ...
 CHECK (Salario>350)
)
```

Restrições de tuplas

- No entanto, ele só funciona bem para restrições a nível de uma única tupla, ou seja, restrições que não precisam de comparações com outras tuplas;
- Inviável para especificar restrições mais complexas:
  - Exemplo: “Nenhum empregado pode ganhar mais do que o seu supervisor”.
- Para estes casos, asserções se tornam uma solução mais prática.

Asserções

- Comandos SQL para a especificações de restrições de integridade;
- Permitem especificar restrições mais complexas que o comando CHECK:
  - Comparação com outras tuplas;
  - Comparação com o resultado de consultas;.
- Uma asserção não é explicitamente amarrada a uma determinada tabela, como o comando CHECK.

Asserções

- A definição de uma asserção em SQL tem a seguinte sintaxe:

  ```
  CREATE ASSERTION Nome_da_Asserção
  CHECK (Restrição_de_Integridade)
  ```

- Geralmente, ao definir uma asserção, selecionamos as possíveis tuplas que violem a restrição de integridade e usamos a cláusula 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 do que o seu supervisor”;
  - Podemos especificá-la através da seguinte asserção:

```
CREATE ASSERTION Limite_Salario
CHECK
(NOT EXISTS
  ( SELECT * FROM Empregado E, Empregado S
    WHERE E.Superivsor=S.Matricula AND E.Salario>S.Salario)
 )
```

Asserções

- Veja que, 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 da consulta não seja vazio, a restrição de integridade foi violada.

Asserções

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

Asserções

- Exemplo 2: Vamos supor a seguinte restrição de integridade: “Nenhum empregado pode trabalhar mais de 30 horas em projetos”;
  - Podemos especificar a restrição através da asserção:

```
CREATE ASSERTION LimiteHoras
CHECK (
    NOT EXISTS (
        SELECT Empregado FROM TrabalhaProjeto TP
        GROUP BY Empregado
        HAVING SUM(NumHoras)>30)
)
```

Asserções

- Exemplo 3: Vamos supor a seguinte restrição de integridade: “Nenhum empregado pode trabalhar em mais de três projetos”;
  - Podemos especificar a restrição através da asserção;

```
CREATE ASSERTION LimiteProjetos
CHECK (
    NOT EXISTS (
        SELECT Empregado FROM TrabalhaProjeto TP
        GROUP BY Empregado
        HAVING COUNT(*)>3)
)
```

Asserções

- Excluindo uma asserção:
  - Podemos excluir uma asserção usando o comando DROP;
    - Sintaxe: DROP ASSERTION Nome_da_Asserção
    - Exemplo: DROP ASSERTION LimiteHoras;

Asserções

- Asserções provocam um grande overhead ao SGBD:
  - Principalmente quando muitos usuários podem atualizar o banco de dados simultaneamente.

Procedimentos Armazenados

- São subprogramas escritos para realizar uma tarefa específica:
  - Semelhante ao conceito de subprogramas em linguagens de programação.
- São armazenados de forma persistente no catálogo do SGBD;
- Embora conhecidos como stored procedures, podem ser procedimentos ou funções;
- Podem ser executados pelo SGBD no momento de uma consulta.

Procedimentos Armazenados

- 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.
  - Diminui o tráfego de dados na rede:
    - Apenas os dados já processados circulam na rede.

Procedimentos Armazenados

- Sintaxe para a criação de funções:
  - CREATE FUNCTION Nome_Função (Parâmetros)
    RETURNS Tipo_Retorno AS ‘
  DECLARE
    DeclaraçõesLocais
    RETURN resultado;
  BEGIN
  Corpo_da_Função;
  END ‘ LANGUAGE Linguagem

Procedimentos Armazenados

- 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];**

Procedimentos Armazenados

- 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.

Procedimentos Armazenados

- Declaração de variáveis locais:
  - Onde:
    - Tipo:
      - Define o tipo de valor que será armazenado na variável;
      - O valor deve ser um dos tipos aceitos pelo SGBD.
    - NOT NULL:
      - Identifica que a variável não pode assumir um valor nulo;
      - Caso ela seja usada, um valor default deve ser especificado para a mesma.

Procedimentos Armazenados

- Declaração de variáveis locais:
  - Onde:
    - DEFAULT:
      - Representa o valor default assumido pela variável;
      - É obrigatório caso ela tenha a restrição NOT NULL.
    - :=
      - Usado para atribuir um valor inicial para a variável.

Procedimentos Armazenados

- Declaração de variáveis locais:
  - Onde:
    - Expressão:
      - Expressão usada para atribuir um valor à variável;
      - O resultado da expressão deve ser compatível com o tipo da variável.

Procedimentos Armazenados

- Declaração de variáveis locais:
  - Exemplos de declarações de variáveis locais:

```
DECLARE
        quantidade INTEGER;
    valor NUMERIC(2) DEFAULT 200;
    soma INTEGER := 0;
    pi CONSTANT NUMERIC(2) := 3.14;
    divisao NUMERIC(2) := (5 / 2);
```

Procedimentos Armazenados

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

```
CREATE OR REPLACE FUNCTION ProximoDepartamento()
RETURNS INTEGER AS '
DECLARE
   MaiorCodigo INTEGER;
BEGIN
   SELECT INTO MaiorCodigo MAX(CodDepartamento) FROM Departamento;
   RETURN MaiorCodigo+1;
END
‘ LANGUAGE plpgsql;
```

Gatilhos

- Um gatilho é um tipo especial de procedimento armazenado que executa quando um determinado evento ocorre no banco de dados.
- É composto por três elementos:
  - Eventos;
  - Condição;
  - Ações.

Gatilhos

- 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 é obrigatória.

Gatilhos

- Ações:
  - São as ações que devem ser executadas pelo SGBD caso o evento aconteça e a condição seja verdadeira;
  - Geralmente, é uma seqüência de comandos SQL;
    - Operações em tabelas;
    - Chamadas a procedimentos armazenados;
    - Etc.

Gatilhos

- 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.

Aula 11 - Integridade de Banco de Dados

Dúvidas?

vitoria@crateus.ufc.br