Banco de Dados II

Restrições de Integridade

Restrições de Integridade

- Descrevem condições lógicas que podem ser válidas para um banco de dados ou para uma sequência de transição de estados do banco de dados no sentido de este estar de acordo com o mini-mundo, representando-o fielmente.

Restrições de Integridade

- Devem representar regras válidas do mundo real no contexto de banco de dados.

- Violações de integridade podem ocorrer:
  - Se partes do mini-mundo forem observadas de uma forma errônea, antes de serem representadas no banco de dados.
  - Se estas observações não forem suficientes para descrever a porção a ser modelada.
  - Se não for feita a comunicação correta do mini-mundo para o SGBD. Logo, o banco de dados não corresponderá ao mundo real.

3

Restrições de Integridade

- parte da integridade do banco de dados deve ser garantida pelos usuários ou pelos programas de aplicação.

Conceito de integridade X segurança

4

Critérios para Classificação das Restrições de Integridade

- Relação ao Modelo de Dados
  - implícitas
  - explícitas
  - inerentes

Critérios para Classificação das Restrições de Integridade

- Simples e Múltiplos Estados do Banco de Dados
  - estática
  - transição
  - temporal

Critérios para Classificação das Restrições de Integridade

- Escopo
  - atributos simples (restrições de domínio)
  - múltiplos atributos de uma simples tupla
  - múltiplas tuplas da uma ou de diferentes relações
  - todas as tuplas de uma relação

Critérios para Classificação das Restrições de Integridade

- Causa da Verificação
  - Usualmente, uma operação de DML (inserção, exclusão ou atualização)
  - Outros eventos
    - Por exemplo, restrições temporais ou dinâmicas, eventos sensíveis ao tempo (*time-related events)*

Critérios para Classificação das Restrições de Integridade

- Tempo de Verificação
  - Imediatas
  - Postergadas

Critérios para Classificação das Restrições de Integridade

- Reação na Violação das Restrições de Integridade
  - restritiva
  - corretiva
  - restrições obrigadas (*deontic constraints)*

Forçar as Restrições

- Forçar as restrições com método baseado na **aplicação**
- Forçar as restrições com método baseado no **SGBD**
  - A manutenção é de responsabilidade do SGBD.

Forçar as Restrições

- O SGBD verifica a validade das restrições quando for realizada uma operação de atualização no banco de dados.
  - Subsistema de integridade (*integrity subsystem) ou monitor de integridade (integrity monitor), que é* responsável pela verificação das restrições.
  - o SGBD analisa o código da transação em fase de pré-compilação e insere dentro da própria transação, operações que verificam e reagem a uma possível violação de uma restrição.

Estratégia

- Todos os eventos que podem causar violação nas restrições devem ser verificados (operações de atualização, tentativa de *commit da transação, …) e* são sinalizados para o monitor de integridade.

Estratégia

- As restrições que necessitam ser verificadas são selecionadas, e a condição de cada uma é usada para consultar, no banco de dados, se houve violação à restrição.

- Em cada ocorrência de violação, a ação apropriada de reação é executada.

Reforçar as Restrições

- **Detectar eventos de atualização**
- **Detectar o esforço no tempo**
- **Determinar as restrições a serem verificadas**
- **Generalizar ou refinar as condições de teste**
- **Verificar as restrições**
- **Determinar as ações relevantes**
- **Executar as ações**

Raciocinando sobre Restrições

- **Satisfatoriedade**: o conjunto de restrições de integridade devem ser livres de contradição, ou seja, não se pode permitir conflitos (contrariedade) entre duas ou mais restrições.

- **Redundância**: quando uma restrição de integridade não adiciona nova semântica ou novas restrições ao modelo, é porque, estas já estão subentendidas em outras restrições.

Restrições de Modelo-Relatado

- Relacional
  - domínios dos atributos
  - integridade da entidade(chave primária)
  - restrição de valores nulos
  - integridade referencial

Regras ECA

- Consistem de:
  - Eventos
  - Condições
  - Ações

- as regras, em SGBD relacional, que atualizam somente uma tupla ou registro são as regras orientadas à instância.

- As regras que atualizam um conjunto de instâncias são chamadas de regras orientadas a conjunto.

Regras ECA – Eventos

- Modificações de dados: insert, delete ou update.
- Recuperação de dados: uma operação select.
- Tempo:
  - Absoluto (por exemplo, 07 Ago 2017 at 01:00)
  - Cíclico (every day at 18:00)
  - ou ainda a intervalos de tempo (every 30 minutes)
- definidos pela aplicação por exemplo, o user-login.

Regras ECA – Condições

- **Predicados no banco de dados:** Por exemplo, nos sistemas de banco de dados relacional, a cláusula *WHERE de SQL.*
- **Consultas ao banco de dados:** Por exemplo, consultas que possuem o resultado verdadeiro ou não vazio.
- **Procedimentos:** a condição pode ser uma chamada a um procedimento escrito na linguagem da aplicação. É indiferente o acesso ou não ao banco de dados do procedimento.

Regras ECA – Ações

- Operações de modificações de dados.
- Operações de recuperação de dados.
- Outros comandos: comandos de controle de transação, como *commit ou rollback.*
- Procedimentos: a ação pode invocar procedimentos escritos na linguagem da aplicação. Não há necessidade do procedimento acessar o banco de dados.

21

Triggers

- Os *triggers seguem o paradigma das* regras ECA.
- O evento é monitorado por uma operação no banco de dados.
- A condição é um predicado SQL arbitrário.

22

Triggers

- A ação é uma sequência de comandos SQL. Porém, comandos que referenciam a transação e a seção do usuário não são permitidos.
- Porém, comandos que referenciam a transação e a seção do usuário não são permitidos.

Triggers

- Triggers são chamadas automaticamente quando os dados da tabela a qual ela esta conectada são alterados.
- Triggers não tem parâmetros de entrada.
- Triggers não retornam valores.
- Triggers são criados pelo comando CREATE TRIGGER.

Triggers

- A diferença entre assertivas e *triggers é* que o comando "ASSERT" deve proibir uma atualização que viola a condição da assertiva, tornando o valor falso.

- Um *trigger permite que a atualização* aconteça e depois executa um procedimento de ação quando ocorre uma violação, pois o valor da condição é verdadeiro.

Triggers

- O comando CREATE TRIGGER, a seguir, mostra todos os elementos da sintaxe do comando.

- As palavras-chave CREATE TRIGGER são seguidas do nome do *trigger, a seguir* a palavra-chave FOR e então o nome da tabela a qual o *trigger estará relacionado.*

- *Em* seguida vem a palavra ACTIVE (ativa) ou INACTIVE (inativa) indicando se o *trigger deverá* ou não ser executado. Se o *trigger está inativo ele não será executado.*

Triggers

- *O próximo elemento do comando* CREATE TRIGGER indica quando o *trigger será executada:*
  - BEFORE UPDATE (Antes de uma atualização)
  - AFTER UPDATE (Após uma atualização)
  - BEFORE INSERT (Antes de uma inclusão)
  - AFTER INSERT (Após uma inclusão)
  - BEFORE DELETE (Antes de uma exclusão)
  - AFTER DELETE (Após uma exclusão)

Triggers

- A seguir vem a palavra chave opcional POSITION seguida de um número inteiro.
- O Firebird permite que você conecte quantas *trigger quiser ao mesmo evento.*
  - *Por exemplo, você* poderia ter quatro *triggers ligadas a tabela EMPLOYEE todas como AFTER UPDATE. Esta é* uma grande característica já que permite que você modularize seu código.
  - Entretanto a ordem em que as *trigger vão ser executadas pode eventualmente ser importante.*
  - *A palavra* chave POSITION te dá o controle da ordem de execução baseado no inteiro informado. No exemplo abaixo a *trigger mostrada será executada primeiro porque a sua posição é 0 (zero).*
  - Se existissem três ou mais *triggers você poderia atribuir as suas posições os valores 10, 20 e* 30. É uma boa ideia deixar um espaço entre a numeração para que você possa facilmente inserir outras *triggers com pontos de execução entre as já criadas.*

Triggers

```
SET TERM^;

CREATE TRIGGER T_SALVA_SALARIO FOR FUNCIONARIOS
ACTIVE AFTER UPDATE
POSITION 0
AS
BEGIN
        IF (old.salario <> new.salario) THEN
                INSERT INTO HISTORICO_SALARIO
                (ID_FUNC, DATA_REAJ, ID_USER, SALARIO_ANT,
PERCENTUAL_REAJ)
                VALUES (old.ID_FUNC,now,new.ID_USER,old.salario,
                        (new.salario - old.salario) * 100 / old.salario);
END^

SET TERM;^
```


Triggers

- Após a palavra chave AS vem a declaração de qualquer variável local usando o comando DECLARE VARIABLE.
- *Finalmente vem o* corpo da *procedure delimitado pelos comandos BEGIN-END.*
- Uma coisa para se ter em mente quando estiver usando *triggers é que uma simples* alteração em um banco de dados pode causar o disparo de vários *triggers.*

Triggers

- Exemplo:
  - Uma alteração na tabela A pode disparar uma *trigger que atualiza a tabela B.*
  - *A atualização da tabela B, por* sua vez, pode disparar uma *trigger que insere um novo registro na tabela C o que pode* provocar o disparo de uma *trigger que atualiza a tabela D e assim sucessivamente.*

Triggers

- O segundo ponto importante sobre *triggers é que um trigger é parte da transação que a* disparou. Isto significa que se você inicia uma transação e atualiza uma linha que dispara um *trigger e este trigger atualiza outra tabela que dispara outro trigger que atualiza outra* tabela e você então dá um ROLLBACK na transação, tanto sua alteração quanto todas as alterações que foram feitas pela série de disparos de *triggers, serão canceladas.*

Triggers - Eventos

- Pode-se ter 6 tipos de eventos em triggers divididos em 2 grupos:
  - Before
    - Insert
    - Update
    - Delete
  - After
    - Insert
    - Update
    - Delete

Triggers - Eventos

- Before:
  - Um *trigger tem que deve ser disparada antes do registro ser atualizado, caso você queira alterar o* valor de uma ou mais colunas antes que a linha seja atualizada ou caso você queira bloquear a alteração da linha gerando uma *EXCEPTION.*
  - *Por exemplo, você teria de usar um trigger* BEFORE DELETE para evitar que o usuário exclua o registro de um cliente que tenha comprado nos últimos dois anos.

Triggers - Eventos

- After:
  - São usados quando você quer garantir que a atualização que disparou a *trigger esteja completa com sucesso antes de você executar outras ações.*

Triggers – OLD e NEW

- No corpo de um *trigger o Firebird deixa disponíveis tanto o valor antigo* como o novo valor de qualquer coluna, por exemplo old.salario e new.salario.

- Usando os valores OLD e NEW você pode facilmente criar registros de histórico, calcular o percentual de alteração de um valor numérico, encontrar em outras tabelas registros que combinem com o valor antigo ou novo de um campo ou fazer qualquer outra coisa que você precise fazer.

Exceptions

- Em um *trigger do tipo BEFORE você pode evitar que a alteração que disparou a trigger seja* efetivada, gerando uma EXCEPTION.

- Antes que você possa gerar uma EXECPTION você precisa cria-la usando o comando CREATE EXCEPTION. Por exemplo:

```
CREATE EXCEPTION E_SALARIO
'Valor inválido.'
```


Exceptions

- Chamada da exceção dento de um trigger:

```
EXCEPTION E_SALARIO;
```

- Quando você gera uma EXCEPTION a execução do *trigger é* terminada.

Generators

- O Firebird não tem um tipo de campo autoincrementável. Ao invés disso tem uma ferramenta mais flexível chamada GENERATOR.
- Um GENERATOR retorna um valor incrementado toda vez que você o chama.
- Para criar um GENERATOR use o comando CREATE GENERATOR como a seguir:

```
CREATE GENERATOR G_INC_FUNC;
```

39

Generators

- Para obter o próximo valor de um GENERATOR use a função GEN_ID() por exemplo:

```
GEN_ID(G_INC_FUNC,1);
```

- Onde 1 é o número do incremento.

40

Exemplo 1

- Criar triggers para restringir que campos de valor nulo (identificados por você) sejam inseridos nas tabelas.

```
Create exception E_VALOR_NULO 'Valor nulo.';
Create trigger T_VALORES_NULOS_CLI for CLIENTE
Active before insert position 1
As
Begin
    If ((new.CPF is null) or (new.DATA_NASC is null)) then
        Exception E_VALOR_NULO;
end
```


Exemplo 2

- Fazer um trigger que impeça a exclusão de um cliente com vendas associadas.

```
Create exception E_IMPEDE 'Registro com outras tabelas associadas.';
Create trigger T_IMPEDE_CLIENTE for CLIENTE
Active before delete position 1
As
Begin
    If (exists(select v.ID_VENDA from VENDA v
            Where v.ID_CLIENTE=old.ID_CLIENTE)) then
        Exception E_IMPEDE;
end
```

Exemplo 3

- Fazer um trigger que calcule o total de uma venda.

```
CREATE OR ALTER TRIGGER T_CALC_VENDA FOR VENDA
ACTIVE BEFORE INSERT OR UPDATE POSITION 5
AS
begin
  new.total=new.subtotal-new.desconto;
end
```


Exemplo 4

- Criar um trigger que ao inserir (ou atualizar) um Item da venda, calcule o somatório dos totais de itens da mesma venda e atualize o sub_total da venda. O campo FINALIZADA deverá ser levado em consideração: “caso seja ‘S’ não poderá atualizar a venda e uma exceção é disparada.”

Exemplo 4 (resolução)

```
CREATE OR ALTER TRIGGER T_ITEMVENDA_INSERIDO
FOR ITEMVENDA

ACTIVE AFTER INSERT OR UPDATE POSITION 1

as

   declare variable finalizada char(1);

   declare variable subtotal numeric(15,2);

begin

   select first 1 v.finalizada

   from venda v

   where v.codvenda = new.codvenda

   into :finalizada;

   if(:finalizada = 'N') then

   begin

      select sum(iv.total)

      from itemvenda iv

      where iv.codvenda = new.codvenda

      into :subtotal;


      update venda v

      set v.subtotal = :subtotal, v.total = :subtotal - v.desconto

      where v.codvenda = new.codvenda;

   end

   else

   begin

      exception e_venda_finalizada;

   end

end
```


Exemplo 5

- Criar um trigger que ao inserir ou editar um item da venda, calcule seu total. Somente poderá ser permitido caso a venda não tenha sido finalizada.

Exemplo 5 (resolução)

```
CREATE OR ALTER TRIGGER T_ITEMVENDA_INSERE FOR ITEMVENDA
ACTIVE BEFORE INSERT OR UPDATE POSITION 1
as
  declare variable finalizada char(1);
begin
  select first 1 v.finalizada
  from venda v
  where v.codvenda = new.codvenda
  into :finalizada;

  if(:finalizada = 'N') then
  begin
            New.TOTAL=new.QUANTIDADE*New.VALOR_UNITARIO;
  end
  else
  begin
    exception e_venda_finalizada;
  end
end
```