Elmasri • Navathe

Sistemas de banco de dados

6ª Edição

PEARSON

Companion Website

A Seção 26.4 é dedicada a conceitos de banco de dados de multimídia. Os **bancos de dados de multimídia** oferecem recursos que permitem que os usuários armazenem e consultem diferentes tipos de informações de multimídia, que incluem **imagens** (como figuras e desenhos), **clipes de vídeo** (como filmes, curtas-metragens e vídeos caseiros), **clipes de áudio** (como músicas, mensagens telefônicas e discursos) e **documentos** (como livros e artigos). Discutimos a análise automática de imagens, o reconhecimento de objeto em imagens e a marcação semântica de imagens.

Na Seção 26.5, discutimos sobre bancos de dados dedutivos,[1] uma área que está na interseção de bancos de dados, lógica e inteligência artificial ou bases de conhecimento. Um **sistema de banco de dados dedutivo** inclui capacidades para definir **regras (dedutivas)**, que podem deduzir informações adicionais dos fatos que estão armazenados em um banco de dados. Como parte da base teórica para alguns sistemas de banco de dados dedutivos é a lógica matemática, essas regras costumam ser chamadas de **bancos de dados lógicos**. Outros tipos de sistemas, conhecidos como **sistemas de banco de dados especialistas** ou **sistemas baseados em conhecimento**, também incorporam capacidades de raciocínio e dedução. Eles utilizam técnicas que foram desenvolvidas no campo da inteligência artificial, incluindo redes semânticas, frames, sistemas de produção ou regras para capturar conhecimento específico do domínio. No final do capítulo há um resumo.

Os leitores podem examinar cuidadosamente os tópicos em que possuem interesse particular, pois as seções deste capítulo são praticamente independentes uma da outra.

26.1 Conceitos de banco de dados ativo e triggers

As regras que especificam ações que são disparadas automaticamente por certos eventos têm sido consideradas melhorias importantes para os sistemas de banco de dados há muito tempo. De fato, o conceito de **triggers** — uma técnica para especificar certos tipos de regras ativas — já existia nas primeiras versões da especificação SQL para bancos de dados relacionais, e as triggers agora fazem parte do padrão SQL-99 e outros mais recentes. Os SGBDs relacionais comerciais — como Oracle, DB2 e Microsoft SQL Server — possuem diversas versões de triggers à disposição. Contudo, desde que os primeiros modelos de triggers foram propostos, muita pesquisa tem sido feita sobre como deve ser um modelo geral para bancos de dados ativos. Na Seção 26.1.1, apresentaremos os conceitos gerais que foram propostos para especificar regras para bancos de dados ativos. Usaremos a sintaxe do SGBD relacional comercial do Oracle para ilustrar esses conceitos com exemplos específicos, pois as triggers do Oracle são próximas ao modo como as regras são especificadas no padrão SQL. A Seção 26.1.2 discutirá algumas questões gerais de projeto e implementação para os bancos de dados ativos. Mostramos exemplos de como os bancos de dados ativos são implementados no SGBD experimental STARBURST na Seção 26.1.3, pois o STARBURST provê muitos dos conceitos de bancos de dados ativos generalizados em sua estrutura. A Seção 26.1.4 discute as aplicações possíveis dos bancos de dados ativos. Finalmente, a Seção 26.1.5 descreve como as triggers são declaradas no padrão SQL-99.

26.1.1 Modelo generalizado para bancos de dados ativos e triggers no Oracle

O modelo que tem sido usado para especificar regras de banco de dados ativo é conhecido como modelo **Evento-Condição-Ação (ECA)**. Uma regra no modelo ECA tem três componentes:

1. O(s) **evento(s)** que dispara(m) a regra: esses eventos normalmente são operações de atualização do banco de dados que são aplicadas explicitamente ao banco de dados. No entanto, no modelo geral, eles também poderiam ser eventos temporais[2] ou outros tipos de eventos externos.
2. A **condição** que determina se a ação da regra deve ser executada: quando o evento que dispara a ação tiver ocorrido, uma condição *opcional* pode ser avaliada. 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.
3. A **ação** a ser tomada: a ação normalmente é uma sequência de comandos SQL, mas também poderia ser uma transação do banco de dados ou um programa externo que será executado automaticamente.

Vamos considerar alguns exemplos para ilustrar esses conceitos. Os exemplos são baseados em uma variação muito simplificada da aplicação de banco

[1] A Seção 26.5 é um resumo dos bancos de dados dedutivos.

[2] Um exemplo seria um evento temporal especificado como uma hora qualquer, como: dispare esta regra todo dia às 17h30.

de dados EMPRESA da Figura 3.5 e que aparece na Figura 26.1, com cada funcionário tendo um nome (Nome), número de cadastro de pessoa física (Cpf), salário (Salario), departamento ao qual eles estão atualmente designados (Dnr, uma chave estrangeira para DEPARTAMENTO) e um supervisor direto (Cpf_supervisor, uma chave estrangeira recursiva para FUNCIONARIO). Para este exemplo, vamos supor que NULL seja permitido para Dnr, indicando que um funcionário pode não estar temporariamente designado a nenhum departamento. Cada departamento tem um nome (Dnome), número (Dnr), o salário total de todos os funcionários designados para o departamento (Sal_total) e um gerente (Cpf_gerente, que é uma chave estrangeira para FUNCIONARIO).

Observe que o atributo Sal_total é, na realidade, um atributo derivado, cujo valor deve ser a soma dos salários de todos os funcionários que estão atribuídos ao departamento em particular. A manutenção do valor correto desse atributo derivado pode ser feita por uma regra ativa. Primeiro, temos de determinar os **eventos** que *podem causar* uma mudança no valor de Sal_total, que são os seguintes:

1. Inserir (uma ou mais) tuplas de novos funcionários.
2. Alterar o salário de (um ou mais) funcionários existentes.
3. Alterar a designação dos funcionários existentes de um departamento para outro.
4. Excluir (uma ou mais) tuplas de funcionários.

No caso do evento 1, só precisamos recalcular Sal_total se o novo funcionário for imediatamente atribuído a um departamento — ou seja, se o valor do atributo Dnr para a nova tupla de funcionário não for NULL (supondo que NULL seja permitido para Dnr). Logo, esta seria a **condição** a ser verificada. Uma condição semelhante poderia ser verificada para o evento 2 (e 4) para determinar se o funcionário cujo salário é alterado (ou que está sendo excluído) está atualmente atribuído a um departamento. Para o evento 3, sempre executaremos uma ação para manter o valor de Sal_total corretamente, de modo que nenhuma condição seja necessária (a ação sempre é executada).

A **ação** para os eventos 1, 2 e 4 é atualizar automaticamente o valor de Sal_total para o departamento do funcionário, a fim de refletir o salário do funcionário recém-inserido, atualizado ou excluído. No caso do evento 3, uma ação dupla é necessária: uma é atualizar o Sal_total do antigo departamento do funcionário e a outra é atualizar o Sal_total do novo departamento do funcionário.

As quatro regras ativas (ou triggers) R1, R2, R3 e R4 — correspondentes à situação acima — podem ser especificadas na notação do SGBD Oracle, como mostra a Figura 26.2(a). Vamos considerar a regra R1 para ilustrar a sintaxe da criação de triggers em Oracle.

A instrução CREATE TRIGGER especifica o nome de uma trigger (ou regra ativa) — Sal_total1 para R1. A cláusula AFTER especifica que a regra será disparada *depois* que ocorrerem os eventos que disparam a regra. Os eventos de disparo — uma inserção de um novo funcionário, neste exemplo — são especificados após a palavra-chave AFTER.[3]

A cláusula ON determina a relação em que a regra é especificada — FUNCIONARIO para R1. As palavras-chave *opcionais* FOR EACH ROW especificam que a regra será disparada *uma vez para cada linha* que é afetada pelo evento de disparo.[4]

A cláusula *opcional* WHEN é utilizada para especificar quaisquer condições que precisam ser verificadas após a regra ser disparada, mas antes que a ação seja executada. Por fim, as ações a serem tomadas são especificadas como um bloco PL/SQL, que normalmente contém um ou mais comandos SQL ou chamadas para executar procedimentos externos.

As quatro triggers (regras ativas) R1, R2, R3 e R4 ilustram uma série de recursos das regras ativas. Primeiro, os **eventos** básicos que podem ser especificados para disparar as regras são os comandos de atualização da SQL padrão: **INSERT**, **DELETE** e **UPDATE**. Eles são especificados pelas palavras-chave **INSERT**, **DELETE** e **UPDATE** na notação Oracle. No caso de **UPDATE**, podem-se especificar os atributos a serem atualizados — por exemplo, ao escrever **UPDATE OF** Salario, Dnr. Segundo, o projetista da regra precisa ter um modo de se referir às tuplas que foram inseridas,

FUNCIONARIO

Nome	Cpf	Salario	Dnr	Cpf_supervisor

DEPARTAMENTO

Dnome	Dnr	Sal_total	Cpf_gerente

Figura 26.1
Um banco de dados EMPRESA simplificado usado para os exemplos de regra ativa.

[3] Conforme veremos, também é possível especificar BEFORE em vez de AFTER, que indica que a regra é disparada *antes que o evento de disparo seja executado*.

[4] Novamente, veremos que uma alternativa é disparar a regra *apenas uma vez*, mesmo que várias linhas (tuplas) sejam afetadas pelo evento de disparo.

```
(a) R1:  CREATE TRIGGER Sal_total1
         AFTER INSERT ON FUNCIONARIO
         FOR EACH ROW
         WHEN ( NEW.Dnr IS NOT NULL )
                 UPDATE DEPARTAMENTO
                 SET Sal_total = Sal_total + NEW.Salario
                 WHERE Dnr = NEW.Dnr;

    R2:  CREATE TRIGGER Sal_total2
         AFTER UPDATE OF Salario ON FUNCIONARIO
         FOR EACH ROW
         WHEN ( NEW.Dnr IS NOT NULL )
                 UPDATE DEPARTAMENTO
                 SET Sal_total = Sal_total + NEW.Salario – OLD.Salario
                 WHERE Dnr = NEW.Dnr;

    R3:  CREATE TRIGGER Sal_total3
         AFTER UPDATE OF Dnr ON FUNCIONARIO
         FOR EACH ROW
                 BEGIN
                 UPDATE DEPARTAMENTO
                 SET Sal_total = Sal_total + NEW.Salario
                 WHERE Dnr = NEW.Dnr;
                 UPDATE DEPARTAMENTO
                 SET Sal_total = Sal_total – OLD.Salario
                 WHERE Dnr = OLD.Dnr;
                 END;

    R4:  CREATE TRIGGER Sal_total4
         AFTER DELETE ON FUNCIONARIO
         FOR EACH ROW
         WHEN ( OLD.Dnr IS NOT NULL )
                 UPDATE DEPARTAMENTO
                 SET Sal_total = Sal_total – OLD.Salario
                 WHERE Dnr = OLD.Dnr;

(b) R5:  CREATE TRIGGER Informar_supervisor1
         BEFORE INSERT OR UPDATE OF Salario, Cpf_supervisor
                 ON FUNCIONARIO
         FOR EACH ROW
         WHEN ( NEW.Salario > ( SELECT Salario FROM FUNCIONARIO
                                WHERE Cpf = NEW.Cpf_supervisor ) )
                 informar_supervisor (NEW.Cpf_supervisor, NEW.Cpf );
```

Figura 26.2

Especificando regras ativas como triggers na notação do Oracle. (a) Triggers para manter automaticamente a consistência de Sal_total de DEPARTAMENTO. (b) Trigger para comparar o salário de um funcionário com o de seu supervisor.

excluídas ou modificadas pelo evento de disparo. As palavras-chave **NEW** e **OLD** são empregadas na notação Oracle; NEW é utilizada para se referir a uma tupla recém-inserida ou recém-atualizada, enquanto OLD é usada para se referir a uma tupla excluída ou a uma tupla antes que ela seja atualizada.

Assim, a regra R1 é disparada após uma operação INSERT ser aplicada à relação FUNCIONARIO. Em R1, a condição (**NEW**.Dnr **IS NOT NULL**) é verificada, e, se for avaliada como verdadeira, significando que a tupla de funcionário recém-inserida está relacionada a um departamento, então a ação é executada. A ação atualiza a(s) tupla(s) de DEPARTAMENTO relacionada(s) ao funcionário recém-inserido, acrescentando seu salário (**NEW**.Salario) ao atributo Sal_total de seu departamento relacionado.

A regra R2 é semelhante a R1, mas é disparada por uma operação UPDATE que atualiza o SALARIO de um funcionário, em vez de por um INSERT. A regra R3 é disparada por uma atualização no atributo Dnr de FUNCIONARIO, o que significa alterar a designação de um funcionário de um departamento para outro. Não existe condição a verificar em R3, de modo que a ação é executada sempre que o evento de disparo ocorre. A ação atualiza tanto o departamento antigo quanto o novo dos funcionários redesignados, somando seu salário a Sal_total de seu *novo* departamento e subtraindo seu salário do Sal_total de seu *antigo* departamento. Observe que isso deve funcionar mesmo que o valor de Dnr seja NULL, pois nesse caso nenhum departamento será selecionado para a ação da regra.[5]

É importante notar o efeito da cláusula FOR EACH ROW, que significa que a regra é disparada separadamente *para cada tupla*. Isso é conhecido como uma **trigger de nível de linha**. Se essa cláusula fosse omitida, a trigger seria conhecida como uma **trigger em nível de comando**, e seria disparada uma vez para cada comando de disparo. Para ver a diferença, considere a seguinte operação de atualização, que gera um aumento de 10 por cento para todos os funcionários designados para o departamento 5. Essa operação seria um evento que dispara a regra R2:

```
UPDATE   FUNCIONARIO
SET      Salario = 1,1 * Salario
WHERE    Dnr = 5;
```

Como o comando acima poderia atualizar vários registros, uma regra que usa a semântica em nível de linha, como R2 na Figura 26.2, seria disparada *uma vez para cada linha*, enquanto uma regra que utiliza a semântica em nível de comando é disparada *apenas uma vez*. O sistema Oracle permite que o usuário escolha qual dessas opções deve ser usada para cada regra. A inclusão da cláusula opcional FOR EACH ROW cria uma trigger em nível de linha, e omiti-la cria uma trigger em nível de comando. Observe que as palavras-chave NEW e OLD só podem ser utilizadas com triggers em nível de linha.

Como um segundo exemplo, suponha que queiramos verificar sempre se o salário de um funcionário é maior que o salário de seu supervisor direto. Vários eventos podem disparar essa regra: inserir um novo funcionário, alterar o salário de um funcionário ou alterar o supervisor de um funcionário. Suponha que a ação a tomar seja chamar um procedimento externo informar_supervisor,[6] que notificará o supervisor. A regra poderia, então, ser escrita como em R5 (ver Figura 26.2(b)).

A Figura 26.3 mostra a sintaxe para especificar algumas das principais opções disponíveis nas triggers Oracle. Na Seção 26.1.5, descreveremos a sintaxe para as triggers no padrão SQL-99.

26.1.2 Questões de projeto e implementação para bancos de dados ativos

A seção anterior forneceu uma visão geral de alguns dos principais conceitos para especificar regras ativas. Nesta seção, discutimos algumas questões adicionais referentes à forma como as regras são projetadas e implementadas. A primeira questão está relacionada à ativação, desativação e agrupamento de regras. Além de criar regras, um sistema de banco de dados ativo deve permitir que os usuários *ativem*, *desativem* e *removam* regras ao referir-se a seus nomes de regra. Uma **regra desativada** não será disparada pelo evento de disparo. Esse recurso permite que os usuários seletivamente desativem regras por certos períodos quando elas não forem necessárias. O **comando de ativação** tornará a regra ativa novamente. O **comando de remoção** exclui a regra do sistema. Outra opção é agrupar as regras em **conjuntos de regras** nomeados, de modo que o conjunto inteiro de regras possa ser ativado, desativado ou removido. Também é útil ter um comando que possa disparar uma regra ou conjunto de regras por meio de um comando PROCESS RULES explícito, emitido pelo usuário.

[5] R1, R2 e R4 também podem ser escritas sem uma condição. Porém, pode ser mais eficiente executá-las com a condição, pois a ação não é chamada a menos que seja exigida.

[6] Considerando que um procedimento externo apropriado tenha sido declarado. Esse é um recurso que está disponível na SQL-99 e em padrões posteriores.

```
<trigger>          ::= CREATE TRIGGER <nome trigger>
                       ( AFTER | BEFORE ) <evento trigger> ON <nome tabela>
                       [ FOR EACH ROW ]
                       [ WHEN <condição> ]
                       <ações da trigger > ;
<evento triggering > ::= <evento trigger> {OR <evento trigger> }
< evento trigger >   ::= INSERT | DELETE | UPDATE [ OF <nome coluna> {, <nome coluna> } ]
< acao trigger >     ::= <bloco PL/SQL>
```

Figura 26.3
Um resumo de sintaxe para especificar triggers no sistema Oracle (apenas opções principais).

A segunda questão diz respeito a se a ação disparada deve ser executada *antes*, *depois*, *no lugar de* ou *juntamente com* o evento de disparo. Uma **trigger before** executa a trigger antes de executar o evento que a causou. Ela pode ser usada em aplicações como a verificação de violações de restrição. Uma **trigger after** executa a trigger depois de executar o evento, e pode ser usada em aplicações como a manutenção de dados derivados e monitoramento de eventos e condições específicas. Uma **trigger instead of** executa a trigger em vez de executar o evento, e pode ser utilizada em aplicações como executar atualizações correspondentes em relações da base em resposta a um evento que é uma atualização de uma visão.

Uma questão relacionada é se a ação que está sendo executada deve ser considerada uma *transação separada* ou se deve fazer parte da mesma transação que disparou a regra. Tentaremos categorizar as diversas opções. É importante observar que nem todas as opções podem estar disponíveis para determinado sistema de banco de dados ativo. De fato, a maioria dos sistemas comerciais é *limitada a uma ou duas das opções* que discutiremos em seguida.

Vamos supor que o evento de disparo ocorra como parte da execução de uma transação. Devemos considerar primeiro as diversas opções de como o evento de disparo está relacionado à avaliação da condição da regra. A *avaliação de condição* da regra também é conhecida como **consideração da regra**, pois a ação deve ser executada somente depois de considerar se a condição é avaliada como verdadeira ou falsa. Existem três possibilidades principais para a consideração da regra:

1. **Consideração imediata.** A condição é avaliada como parte da mesma transação que o evento de disparo, e é avaliada *imediatamente*. Esse caso pode ser categorizado ainda em três opções:
   - Avaliar a condição *antes* de executar o evento de disparo.
   - Avaliar a condição *depois* de executar o evento de disparo.
   - Avaliar a condição *em vez de* executar o evento de disparo.
2. **Consideração adiada.** A condição é avaliada ao final da transação que incluiu o evento de disparo. Nesse caso, pode haver muitas regras disparadas esperando para ter suas condições avaliadas.
3. **Consideração separada.** A condição é avaliada como uma transação separada, gerada com base na transação de disparo.

O próximo conjunto de opções refere-se ao relacionamento entre a avaliação da condição de regra e a *execução* da ação da regra. Aqui, novamente, três opções são possíveis: execução **imediata**, **adiada** ou **separada**. A maioria dos sistemas ativos utiliza a primeira opção. Ou seja, assim que a condição é avaliada, se ela retornar verdadeira, a ação é executada *imediatamente*.

O sistema Oracle (ver Seção 26.1.1) utiliza o modelo de *consideração imediata*, mas permite que o usuário especifique para cada regra se a opção *before* ou *after* deve ser usada com a avaliação de condição imediata. Ele também usa o modelo de *execução imediata*. O sistema STARBURST (ver Seção 26.1.3) tem a opção de *consideração adiada*, significando que todas as regras disparadas por uma transação esperam até que a transação de disparo alcance seu fim e emita seu comando COMMIT WORK antes que as condições da regra sejam avaliadas.[7]

[7] O STARBURST também permite que o usuário inicie a consideração da regra explicitamente por meio do comando PROCESS RULES.

Outra questão referente a regras de banco de dados ativo é a distinção entre *regras em nível de linha* e *regras em nível de comando*. Como as instruções de atualização SQL (que atuam como eventos de disparo) podem especificar um conjunto de tuplas, é preciso distinguir se a regra deve ser considerada uma vez para o *comando inteiro* ou se deve ser considerada separadamente *para cada linha* (ou seja, tupla) afetada pelo comando. O padrão SQL-99 (ver Seção 26.1.5) e o sistema Oracle (ver Seção 26.1.1) permitem que o usuário escolha qual das opções deve ser usada para cada regra, enquanto o STARBURST utiliza apenas a semântica em nível de comando. Daremos exemplos de como as triggers em nível de comando podem ser especificadas na Seção 26.1.3.

Uma das dificuldades que podem ter limitado o uso generalizado de regras ativas, apesar de seu potencial para simplificar o desenvolvimento de banco de dados e software, é que não existem técnicas de fácil utilização para projetar, escrever e verificar regras. Por exemplo, é muito difícil verificar se um conjunto de regras é **consistente**, significando que duas ou mais regras no conjunto não contradizem uma à outra. É difícil garantir o **término** de um conjunto de regras sob todas as circunstâncias. Para ilustrar o problema do término resumidamente, considere as regras da Figura 26.4. Aqui, a regra R1 é disparada por um evento INSERT na TABLE1 e sua ação inclui um evento de atualização em Attribute1 de TABLE2. Porém, o evento de disparo da regra R2 é um evento UPDATE em Attribute1 de TABLE2, e sua ação inclui um evento INSERT na TABLE1. Neste exemplo, é fácil ver que essas duas regras podem disparar uma à outra indefinidamente, levando ao não término. Contudo, se dezenas de regras forem escritas, é muito difícil determinar se o término é garantido ou não.

```
R1:CREATE TRIGGER T1
    AFTER INSERT ON TABLE1
    FOR EACH ROW
            UPDATE TABLE2
            SET Attribute1 = ... ;
R2:CREATE TRIGGER T2
    AFTER UPDATE OF Attribute1 ON TABLE2
    FOR EACH ROW
            INSERT INTO TABLE1 VALUES ( ... );
```

Figura 26.4
Um exemplo para ilustrar o problema de término para regras ativas.

Se as regras ativas tiverem de alcançar seu potencial, é necessário desenvolver ferramentas para o projeto, depuração e monitoramento de regras ativas que possam ajudar os usuários a projetarem e depurarem suas regras.

26.1.3 Exemplos de regras ativas em nível de comando no STARBURST

Agora, oferecemos alguns exemplos para ilustrar como as regras podem ser especificadas no SGBD experimental STARBURST. Isso nos permitirá demonstrar como as regras em nível de comando podem ser escritas, pois estes são os únicos tipos de regras permitidas no STARBURST.

As três regras ativas R1S, R2S e R3S da Figura 26.5 correspondem às três primeiras regras da Figura 26.2, mas elas usam a notação do STARBURST e a semântica em nível de comando. Podemos explicar a estrutura da regra com a regra R1S. O comando CREATE RULE especifica um nome de regra — Sal_total1 para R1S. A cláusula ON especifica a relação na qual a regra é especificada — FUNCIONARIO para R1S. A cláusula WHEN é usada para especificar os **eventos** que disparam a regra.[8] A cláusula IF *opcional* é utilizada para especificar quaisquer **condições** que precisam ser verificadas. Finalmente, usa-se a cláusula THEN para especificar as **ações** a serem tomadas, que normalmente são um ou mais comandos SQL.

No STARBURST, os eventos básicos que podem ser especificados para disparar as regras são os comandos de atualização SQL padrão: INSERT, DELETE e UPDATE. Estes são especificados pelas palavras-chave **INSERTED**, **DELETED** e **UPDATED** na notação do STARBURST. Segundo, o projetista de regra precisa ter um modo de referenciar as tuplas que foram modificadas. As palavras-chave **INSERTED**, **DELETED**, **NEW-UPDATED** e **OLD-UPDATED** são empregadas na notação do STARBURST para se referirem a quatro **tabelas de transição** (relações) que incluem as tuplas recém-inseridas, as tuplas excluídas, as tuplas atualizadas *antes* que fossem atualizadas e as tuplas atualizadas *depois* que foram atualizadas, respectivamente. Obviamente, dependendo dos eventos de disparo, somente algumas dessas tabelas de transição podem estar disponíveis. O escritor da regra pode se referir a essas tabelas ao escrever as partes de condição e ação dela. As tabelas de transição contêm tuplas do mesmo tipo que aquelas na relação especificada na cláusula ON da regra — para R1S, R2S e R3S, esta é a relação FUNCIONARIO.

[8] Observe que a palavra-chave WHEN especifica *eventos* no STARBURST, mas serve para especificar a regra *condição* em SQL e triggers Oracle.

```
R1S:  CREATE RULE Sal_total1 ON FUNCIONARIO
      WHEN INSERTED
      IF   EXISTS   ( SELECT * FROM INSERTED WHERE Dnr IS NOT NULL )
      THEN UPDATE   DEPARTAMENTO AS D
           SET      D.Sal_total = D.Sal_total +
                    ( SELECT SUM (I.Salario) FROM INSERTED AS I WHERE D.Dnr = I.Dnr )
           WHERE    D.Dnr IN ( SELECT Dnr FROM INSERTED );

R2S:  CREATE RULE Sal_total2 ON FUNCIONARIO
      WHEN UPDATED  ( Salario )
      IF   EXISTS   ( SELECT * FROM NEW-UPDATED WHERE Dnr IS NOT NULL )
           OR EXISTS ( SELECT * FROM OLD-UPDATED WHERE Dnr IS NOT NULL )
      THEN UPDATE   DEPARTAMENTO AS D
           SET      D.Sal_total = D.Sal_total +
                    ( SELECT SUM (N.Salario) FROM NEW-UPDATED AS N
                      WHERE D.Dnr = N.Dnr ) -
                    ( SELECT SUM (O.Salario) FROM OLD-UPDATED AS O
                      WHERE D.Dnr = O.Dnr )
           WHERE    D.Dnr IN ( SELECT Dnr FROM NEW-UPDATED ) OR
                    D.Dnr IN ( SELECT Dnr FROM OLD-UPDATED );

R3S:  CREATE RULE Sal_total3 ON FUNCIONARIO
      WHEN UPDATED  ( Dnr )
      THEN UPDATE   DEPARTAMENTO AS D
           SET      D.Sal_total = D.Sal_total +
                    ( SELECT SUM (N.Salario) FROM NEW-UPDATED AS N
                      WHERE D.Dnr = N.Dnr )
           WHERE    D.Dnr IN ( SELECT Dnr FROM NEW-UPDATED );
           UPDATE   DEPARTAMENTO AS D
           SET      D.Sal_total = D.Sal_total -
                    ( SELECT SUM (O.Salario) FROM OLD-UPDATED AS O
                      WHERE D.Dnr = O.Dnr )
           WHERE    D.Dnr IN ( SELECT Dnr FROM OLD-UPDATED );
```

Figura 26.5
Regras ativas usando semântica em nível de comando na notação do STARBURST.

Na semântica em nível de comando, o projetista da regra só pode se referir às tabelas de transição como um todo, e a regra é disparada apenas uma vez, de modo que as regras precisam ser escritas de forma diferente daquela para a semântica em nível de linha. Como várias tuplas de funcionários podem ser inseridas em um único comando de inserção, temos de verificar se *pelo menos uma* das tuplas de funcionário recém-inseridas está relacionada a um departamento. Em R1S, a condição

EXISTS (**SELECT** * **FROM INSERTED WHERE** Dnr **IS NOT NULL**)

é verificada e, se for avaliada como verdadeira, então a ação é executada. A ação atualiza em um único comando as tuplas de DEPARTAMENTO relacionadas aos funcionários recém-inseridos ao acrescentar seus salários ao atributo Sal_total de cada departamento relacionado. Como mais de um funcionário recém-inserido pode pertencer ao mesmo departamento, usamos a função de agregação SUM para garantir que todos os seus salários sejam atualizados.

A regra R2S é semelhante à R1S, mas é disparada por uma operação UPDATE que atualiza o salário de um ou mais funcionários, em vez de um

INSERT. A regra R3S é disparada por uma atualização no atributo Dnr de FUNCIONARIO, que significa alterar a designação de um ou mais funcionários de um departamento para outro. Não existe condição em R3S, de modo que a ação é executada sempre que o evento de disparo ocorre.[9] A ação atualiza tanto o(s) departamento(s) antigo(s) quanto o(s) departamento(s) novo(s) dos funcionários redesignados, somando seu salário a Sal_total de cada departamento *novo* e subtraindo-o de Sal_total de cada departamento *antigo*.

Em nosso exemplo, é mais complexo escrever as regras em nível de comando do que em nível de linha, como pode ser ilustrado ao se comparar as figuras 26.2 e 26.5. No entanto, essa não é uma regra geral, e outros tipos de regras ativas podem ser mais fáceis de especificar quando se usa a notação em nível de comando do que quando se usa a notação em nível de linha.

O modelo de execução para regras ativas no STARBURST usa a **consideração adiada**. Ou seja, todas as regras que são disparadas em uma transação são colocadas em um conjunto — chamado **conjunto de conflito** — que não é considerado para avaliação de condições e execução até que a transação termine (emitindo seu comando COMMIT WORK). O STARBURST também permite que o usuário inicie explicitamente a consideração da regra no meio de uma transação por meio de um comando PROCESS RULES explícito. Como várias regras precisam ser avaliadas, é necessário especificar uma ordem entre as regras. A sintaxe para a declaração da regra no STARBURST permite a especificação da *ordenação* entre as regras para instruir o sistema sobre a ordem em que um conjunto de regras deve ser considerado.[10] Além disso, as tabelas de transição — INSERTED, DELETED, NEW-UPDATED e OLD-UPDATED — contêm o *efeito de entrelaçamento* (net effect) de todas as operações na transação que afetaram cada tabela, pois múltiplas operações podem ter sido aplicadas a cada tabela durante a transação.

26.1.4 Aplicações em potencial para bancos de dados ativos

Agora, discutimos rapidamente algumas das aplicações em potencial das regras ativas. Obviamente, uma aplicação importante é permitir a **notificação** de certas condições que ocorrem. Por exemplo, um banco de dados ativo pode ser usado para monitorar, digamos, a temperatura de uma fornalha industrial. A aplicação pode inserir periodicamente no banco de dados os registros de leitura de temperatura diretamente dos sensores de temperatura, e regras ativas podem ser escritas, que são ativadas sempre que um registro de temperatura for inserido, com uma condição que verifica se a temperatura excede o nível de perigo, e resulta na ação para disparar um alarme.

As regras ativas também podem ser usadas para **impor restrições de integridade** ao especificar os tipos de eventos que podem fazer que as restrições sejam violadas e, depois, avaliar condições apropriadas que verificam se as restrições são realmente violadas pelo evento ou não. Logo, as restrições de aplicação complexas, normalmente conhecidas como **regras de negócios**, podem ser impostas dessa forma. Por exemplo, na aplicação de banco de dados UNIVERSIDADE, uma regra pode monitorar a média dos alunos sempre que uma nova nota for inserida, e pode alertar o conselho se a média de um aluno ficar abaixo de certo patamar. Outra regra pode verificar se os pré-requisitos do curso são satisfeitos antes de permitir que um aluno se matricule em uma disciplina; e assim por diante.

Outras aplicações incluem a **manutenção automática de dados derivados**, como os exemplos das regras de R1 a R4 que mantêm o atributo derivado Sal_total sempre que tuplas de funcionário individual são alteradas. Uma aplicação semelhante é usar regras ativas para manter a consistência de **visões materializadas** (ver Seção 5.3) sempre que as relações da base são modificadas. Como alternativa, uma operação de atualização especificada em uma visão pode ser um evento de disparo, que pode ser convertido para atualizações nas relações de base ao usar uma trigger *instead of*. Essas aplicações também são relevantes para as novas tecnologias de data warehousing (ver Capítulo 29). Uma aplicação relacionada mantém que **tabelas replicadas** são consistentes ao especificar regras que modificam as réplicas sempre que a tabela mestra é modificada.

26.1.5 Triggers na SQL-99

As triggers na SQL-99 e padrões posteriores são muito semelhantes aos exemplos que discutimos na Seção 26.1.1, com algumas pequenas diferenças sintáticas. Os **eventos** básicos que podem ser especificados para disparar as regras são os comandos de atualização SQL padrão: INSERT, DELETE e UPDATE. No caso de UPDATE, podem-se especificar os atributos a serem atualizados. Tanto as triggers em nível de linha quando em nível de comando são permitidas, indica-

[9] Assim como nos exemplos do Oracle, as regras R1S e R2S podem ser escritas sem uma condição. Porém, pode ser mais eficiente executá-las com a condição, já que a ação não é chamada a menos que seja exigida.

[10] Se nenhuma ordem for especificada entre um par de regras, a ordem default do sistema é baseada na colocação da regra declarada primeiro, antes da outra regra.

```
T1: CREATE TRIGGER Sal_total1
    AFTER UPDATE OF Salario ON FUNCIONARIO
    REFERENCING OLD ROW AS O, NEW ROW AS N
    FOR EACH ROW
    WHEN ( N.Dnr IS NOT NULL )
    UPDATE DEPARTAMENTO
    SET Sal_total = Sal_total + N.salario – O.salario
    WHERE Dnr = N.Dnr;

T2: CREATE TRIGGER Sal_total2
    AFTER UPDATE OF Salario ON FUNCIONARIO
    REFERENCING OLD TABLE AS O, NEW TABLE AS N
    FOR EACH STATEMENT
    WHEN  EXISTS ( SELECT * FROM N WHERE N.Dnr IS NOT NULL ) OR
          EXISTS ( SELECT * FROM O WHERE O.Dnr IS NOT NULL )
    UPDATE DEPARTAMENTO AS D
    SET D.Sal_total = D.Sal_total
    + ( SELECT SUM (N.Salario) FROM N WHERE D.Dnr=N.Dnr )
    – ( SELECT SUM (O.Salario) FROM O WHERE D.Dnr=O.Dnr )
    WHERE Dnr IN ( ( SELECT Dnr FROM N ) UNION ( SELECT Dnr FROM O ) );
```

Figura 26.6
Trigger T1 ilustrando a sintaxe para definir triggers na SQL-99.

das na trigger pelas cláusulas FOR EACH ROW e FOR EACH STATEMENT, respectivamente. Uma diferença sintática é que a trigger pode especificar nomes de variável de tupla em particular para as tuplas antiga e nova em vez de usar as palavras-chave NEW e OLD, como mostra a Figura 26.1. A trigger T1 da Figura 26.6 mostra como a trigger em nível de linha R2 da Figura 26.1(a) pode ser especificada na SQL-99. Dentro da cláusula REFERENCING, nomeamos variáveis de tupla (apelidos) O e N para nos referirmos à tupla OLD (antes da modificação) e à tupla NEW (após a modificação), respectivamente. A trigger T2 da Figura 26.6 mostra como a trigger em nível de comando R2S da Figura 26.5 pode ser especificada na SQL-99. Para uma trigger em nível de comando, a cláusula REFERENCING é usada para se referir à tabela de todas as tuplas novas (recém-inseridas ou recém-atualizadas) como N, enquanto a tabela de todas as tuplas antigas (tuplas excluídas ou tuplas antes que sejam atualizadas) é referenciada como O.

26.2 Conceitos de banco de dados temporal

Bancos de dados temporais, no sentido mais amplo, abrangem todas as aplicações de banco de dados que exigem algum aspecto de tempo quando organizam suas informações. Logo, elas oferecem um bom exemplo para ilustrar a necessidade de desenvolver um conjunto de conceitos de unificação para os desenvolvedores de aplicação usarem. Aplicações de banco de dados temporal têm sido desenvolvidas desde os primeiros dias do uso do banco de dados. Porém, na criação dessas aplicações, fica principalmente a cargo dos projetistas e desenvolvedores de aplicação descobrir, projetar, programar e implementar os conceitos temporais de que eles necessitam. Existem muitos exemplos de aplicações em que algum aspecto de tempo é necessário para manter as informações em um banco de dados. Entre eles estão a área de *saúde*, em que históricos de paciente precisam ser mantidos; *seguro*, em que históricos de acicles e sinistros são necessários, bem como informações sobre as datas em que as apólices de seguro estão em vigor; *sistemas de reserva* em geral (hotel, companhia aérea, aluguel de carro etc.), em que informações sobre datas e horários das reservas são necessárias; *bancos de dados científicos*, em que os dados coletados de experimentos incluem o horário em que cada dado é medido; e assim por diante. Até mesmo os dois exemplos usados neste livro podem ser facilmente expandidos para aplicações temporais. No banco de dados EMPRESA, podemos querer manter históricos de SALARIO, CARGO e PROJETO sobre