BANCOS DE DADOS ATIVOS

Mariano Cilia[†]

Universidad Nacional del Centro de la Provincia de Buenos Aires

Facultad de Ciencias Exactas - Instituto de Sistemas Tandil.

San Martin 57, (7000) Tandil, Buenos Aires, Argentina

e-mail: mcilia@dcc.unicamp.br

SUMÁRIO

Os bancos de dados ativos são sistemas de banco de dados estendidos com um sistema de regras. Este sistema é capaz de reconhecer eventos, ativar as regras correspondentes e quando a condição é verdadeira executa as ações que correspondam, segundo o paradigma E-C-A proposto em [DBM88]. Este sistemas podem ser usados para aplicações financeiras (*commodity trading, portfolio management, currency trading, etc.*), aplicações multimídia, controle da produção industrial (*CIM*, controle de inventario, etc.), monitoramento (controle de tráfego aéreo, etc), entre outros [Buch94]. Também são usados para funções do próprio núcleo do banco de dados, como por exemplo, manutenção de consistência, manutenção de visões, controle de acesso, gerenciamento de versões, entre outras.

Neste artigo descrevem-se as principais características dos bancos de dados ativos que são classificadas em três grupos: definição de regras, modelo de execução e otimização. Logo são descritos as características ativas dos principais protótipos desenvolvidos na área.

1. INTRODUÇÃO

Um banco de dados é *passivo* quando não oferece suporte para o gerenciamento automático de condições definidas sobre o estado do banco de dados em resposta a estímulos externos. Os sistemas de gerenciamento de banco de dados (SGBDs) convencionais são passivos, só executando transações quando são explicitamente requisitadas pelo usuário ou aplicação.

Um banco de dados é *ativo* quando *eventos* gerados interna ou externamente ao sistema provocam uma resposta do próprio banco de dados (BD), independente da solicitação do usuário. Neste caso, alguma *ação* é tomada automaticamente dependendo das *condições* que foram especificadas sobre o estado do banco de dados. Este paradigma

---

[†] Atualmente realizando estudos de posgraduação no Departamento de Ciências da Computação - IMECC - UNICAMP, Campinas, Sõ Paulo, Brasil.

é útil para implementar várias funções do próprio banco de dados ou mesmo para estendê-las. Alguns exemplos de aplicações são: controle de integridade, controle de acesso, políticas de segurança e atualização.

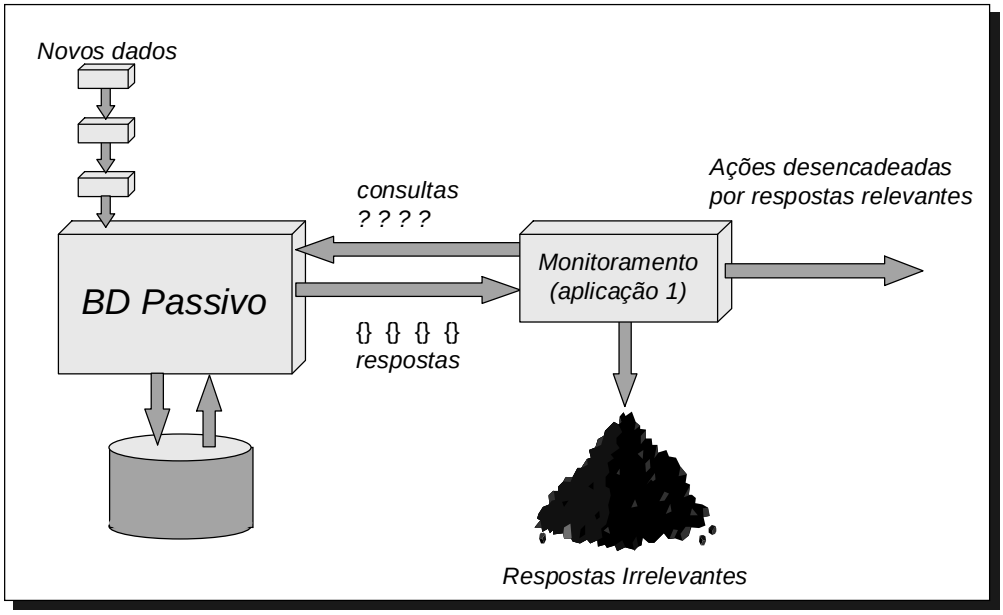

Figura 1. Polling [Buch94].

Há diferentes formas de transformar um banco de dados passivo em ativo. Segundo [CN90], as mais tradicionais são: escrever no próprio programa de aplicação a condição que se deseja testar; e avaliar continuamente a condição, ou seja, polling (figura 1). Uma desvantagem da primeira alternativa é que a avaliação da condição é responsabilidade do programador. No segundo caso, o problema pode ser a baixa utilização dos recursos se houver uma excessiva frequência dos testes de condição. Estes problemas são resolvidos parcialmente com o uso de regras e gatilhos.

Bancos de dados *dedutivos* fornecem um mecanismo para derivar dados que não estão explicitamente armazenados no banco de dados (conhecidos como dados virtuais ou dados derivados). São mais poderosos e expressivos que as visões, embora mais problemáticos para serem suportados [LT92].

Não existe uma divisão óbvia entre os bancos de dados dedutivos e os ativos. A principal diferença está baseada no modelo de execução. No primeiro tipo, geralmente a preocupação é a derivação de informação, e as regras são executadas explicitamente pela aplicação. No segundo, as regras (ou gatilhos) são disparadas como efeito colateral das ações normais do banco de dados [Wid93].

As pesquisas em banco de dados ativos (BDA) podem ser divididas em três categorias: i) a definição das regras ou gatilhos, ii) seu correspondente modelo de execução, e iii) a sua otimização. Na primeira categoria definem-se quais são os tipos de eventos, condições e ações. Uma questão muito importante é a expressividade da linguagem de especificação. Na segunda categoria, a maioria dos artigos especifica com detalhe os modelos de execução e alguns outros só esquematizam o sistema

implementado. Nesta categoria discute-se quando devem ser avaliadas as condições, ou como devem ser resolvidos os conflitos (quando mais de uma regra é habilitada ao mesmo tempo). Por último, existe o problema da otimização da execução, tendo em consideração estratégias eficientes para avaliação da condição.

Na seção seguinte são descritas algumas das aplicações dos sistemas de gerenciamento de BDA. A seguir são apresentadas as características dos BDA segundo as três categorias descritas anteriormente (definição de regras, modelo de execução, e otimização da execução). Finalmente, são descritas as características "ativas" dos principais protótipos desenvolvidos na área.

2. USO DOS SISTEMAS ATIVOS

Os sistemas de bancos de dados ativos podem ser classificados em três grupos segundo seu uso [VK93], a saber:

- ***Suporte automático ao usuário***
    - *Notificação:* Em algumas situações o banco de dados precisa da intervenção do usuário. As regras podem ser usadas para detectar automaticamente estas situações e informar ao usuário.
    - *Execução automática de procedimentos:* Ante a ocorrência de um evento, um procedimento pode ser executado mediante o uso do sistema de regras.
    - *Provimento de valores default:* Uma operação comum nas aplicações é a atribuição de valores *default*, que pode ser feita diretamente com regras.
- ***Funcionalidade do modelo de dados***
    - *Manutenção da integridade:* Uma *restrição de integridade*, num ambiente de BD, é um predicado sobre estados do BD que deve ser mantida, para assegurar a consistência dos dados. Num sistema ativo, as condições que violam a integridade são especificadas na parte da condição da regra e a ação corretora especificada na parte correspondente à ação. Este esquema traz duas vantagens importantes: i) o BD suporta especificação e manutenção de restrições, sem depender do código da aplicação, ii) o BD serve para todas as aplicações que têm a mesma visão do mundo. Maiores detalhes sobre manutenção de integridade podem ser encontrados em [MA92, CW90].
    - *Proteção:* O acesso ao banco de dados pode ser controlado usando regras. Com este esquema não só pode ser aceito ou rejeitado o acesso aos dados, como também o sistema pode fornecer dados fictícios nos campos protegidos.
- ***Gerenciamento dos recursos***

  As regras são usadas dentro do próprio núcleo do BD.

- *Otimização do armazenamento físico:* Podem ser usadas para ações de *checkpoint*, *clustering*, caminhos de acesso, ou também para adaptar as estruturas de armazenamento segundo estatísticas das consultas.

- *Gerenciamento de visões:* Visões materializadas complexas podem ser mantidas com regras [SJGP90].

3. REGRAS E GATILHOS

Bancos de dados ativos são via de regra implementados a partir de mecanismos de regras de produção. Regras são descrições de comportamento a serem adotadas por um sistema. As regras estão geralmente baseadas em três componentes: *evento*, *condição* e *ação* (E-C-A) [DBM88]. O evento é um indicador da ocorrência de uma determinada situação. Uma condição é um predicado sobre o estado do banco de dados. Uma ação é um conjunto de operações a ser executado quando um determinado evento ocorre e a sua condição é avaliada como verdadeira. Um evento pode disparar uma ou mais regras.

O primeiro na formalização de sistemas ativos foi Morgenstein. No trabalho descrito em [Mor84] utilizam-se os sistemas ativos para manter restrições através das equações de restrições (Constraint Equation, CEs). As CEs permitem expressar restrições semânticas que requerem consistência entre muitas relações, de forma similar à manutenção de relações. As CEs constituem uma forma mais concisa que a escrita de procedimentos para expressar e garantir restrições, por possuírem representação declarativa, e uma interpretação executável, sendo compiladas em rotinas para garantir automaticamente as restrições.

De acordo com [SR88], os gatilhos são associações de condições e ações. A execução da ação ocorre quando o banco de dados evolui para um estado que leva o gatilho à condição verdadeira.

3.1. REGRAS E-C-A

A forma atualmente aceita para considerar un BDA é a adoção de regras de produção E-C-A.

- ***Evento.*** O evento é um indicador da ocorrência de uma determinada situação (***quando*** avaliar). Como é mostrado na figura 2 existem basicamente três tipos de eventos: temporais (às 8:30, repetidas vezes toda sexta às 10:00), definidos pelo usuário (alta temperatura, user-login, etc.), e operações próprias dos BD (insert, delete, update, select).

- ***Condição.*** Uma condição é um predicado sobre o estado do banco de dados (***o que*** avaliar). Condições são comumente implementadas por consultas ou por procedimentos da aplicação.

- ***Ação.*** Uma ação é um conjunto de operações a ser executado quando um determinado evento ocorre e a condição associada é avaliada como verdadeira (***como***

responder). Um evento pode disparar uma ou mais regras. As ações típicas são: operações de modificação ou consulta, comando do BD (commit, rollback), ou procedimentos da aplicação (podendo ou não acessar o BD).

Existem dois aspectos importantes no projeto da linguagem de regras de produção: a sintaxe para a criação, modificação, e eliminação de regras; e a semântica do processamento das regras na execução. A sintaxe da maioria das linguagens é similar (baseadas na extensão da linguagem de consulta). No entanto, a semântica do processamento varia consideravelmente.

3.2. COMPONENTES

Na figura 2 são mostrados os componentes de um BDA, que adiciona características ativas (que serão descritas a seguir) às funções características dos BD convencionais (controle de concorrência, recuperação ante falhas, persistência, otimização de consultas, e outras [ABD+92]).

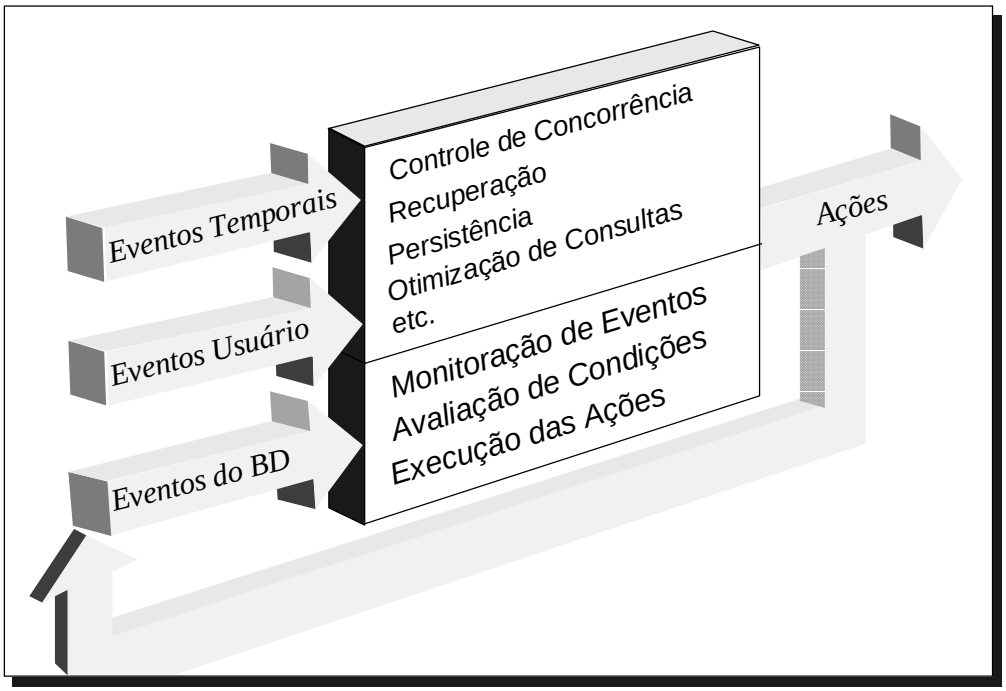

Figura 2. Componentes de um Banco de Dados Ativo [Buch94].

São três as componentes básicas para a obtenção de um sistema ativo:

- ***Monitoramento de Eventos:*** É o módulo encarregado de detectar eventos e ativar as regras que dependam desse evento.

- ***Avaliação da Condição:*** Depois que o evento foi detectado o avaliador da condição é responsável pela avaliação eficiente das condições. Aquelas regras cujas condições sejam verdadeiras são passadas para o *Executor de ações*.

- ***Execução de Ações:*** Este componente coordena o sincronismo entre detecção de eventos e execução de ações. As ações podem ser executadas de imediato (antes do fim da transação que disparou a regra), ou depois (numa transação independente). A ligação entre a execução de ações e regra é denominada *modo de acoplamento*.

Dependendo dos *modos de acoplamento* a serem adotados pelo sistema de regras, o gerenciador de transações subjacente deve suporte transações aninhadas.

3.3. LINGUAGENS PARA ESPECIFICAR EVENTOS

Um BDA precisa detectar a ocorrência de qualquer evento definido para poder iniciar a ativação das regras. Para que as situações reais possam ser monitoradas, uma linguagem de definição de eventos deve ser empregada permitindo a modelagem de situações complexas. Alguns exemplos deste tipo de linguagens são Ode [GSJ92b], Samos [GD93], Compose [GJS92a] e Snoop [CM93] entre outros.

Álgebras de eventos são incorporadas às linguagens de especificação de eventos para permitir a composição de eventos. Os principais operadores encontrados em álgebras de composição como as de Ode, Samos, Snoop, e Compose são:

- ***Seqüência.*** Indica que o evento composto acontece quando os eventos que constituem a seqüência tiverem ocorrido na ordem determinada.
- ***Conjunção.*** Indica que o evento composto acontece se todos eventos que formam a conjunção ocorreram, independente da ordem relativa entre eles.
- ***Disjunção.*** Indica que o evento composto ocorre se pelo menos um dos eventos que formam a disjunção tiver acontecido.
- ***Negação.*** Indica que o evento composto acontece se os eventos descritos na expressão de negação não tiverem acontecido num determinado intervalo de tempo.

3.4. LINGUAGENS PARA ESPECIFICAR CONDIÇÕES

Condições são expressas por fórmulas, às quais é atribuído um valor booleano quando executadas. Uma classificação das possíveis teorias da lógica para linguagens de especificação de condições é descrita em [VK93]. Entre as teorias estão a lógica proposicional, lógica de primeira ordem, cláusulas de Horn, e a lógica temporal.

As linguagens de consulta são geralmente usadas para a especificação de condições. Segundo seu resultado quando avaliada (vazio ou não) a condição será verdadeira dependendo da convenção adotada.

3.5. LINGUAGENS PARA ESPECIFICAR AÇÕES

As linguagens para especificação de ações podem ser divididas em três grupos: linguagens de consulta, linguagens de consulta estendidas, e acesso direto ao banco de dados.

- ***Linguagens de consulta:*** As ações são especificadas com a mesma linguagem de consulta dos sistemas de bancos de dados, como por exemplo SQL. A vantagem é que o acesso ao banco de dados fica sob controle do próprio sistema, e a desvantagem é que limita a funcionalidade. Por exemplo, não podem ser executadas ações externas ao ambiente do banco de dados.

- ***Linguagens de consulta estendidas:*** Para que as regras possam interagir com o ambiente externo, a linguagem de consulta precisa ser estendida. Assim a linguagem de especificação de ações pode basear-se em duas partes, a linguagem de consulta e a linguagem de comunicação. Esta última é baseada em chamadas a procedimentos convencionais ou em primitivas *send* e *receive*.

- ***Acesso direto ao banco de dados:*** Uma linguagem algorítmica aumenta a expressividade das ações, quando não podem ser expressas com uma linguagem de consulta estendida. A desvantagem deste tipo de linguagem é a redução das possibilidades de otimização, que é de vital importância para obter uma performance aceitável.

4. MODELOS DE EXECUÇÃO

Os sistemas de banco de dados tradicionais são passivos. A inclusão de gatilhos (ou regras) os convertem em sistemas ativos, com a caraterística de executar automaticamente tais gatilhos (ou regras). Esta inclusão afeta significativamente o modelo de execução.

Segundo [VK93] os conceitos mais importantes envolvidos são:

- ***Granularidade da ativação:*** É o nível onde a ativação do gatilho é detectada. Existem quatro opções para a escolha da granularidade: a sessão do próprio banco de dados (A1), a transação (A2), o comando (A3) e a operação (ou primitiva) do banco de dados (A4). Isto significa que a ativação dos gatilhos ou regras só pode ser feita respectivamente entre sessões, transações, comandos e operações.

- ***Granularidade dos gatilhos (ou regras):*** Representa a atomicidade de execução do gatilho (ou regra), a saber: a própria regra é a unidade de execução (T1); evento, condição e ação são individualmente atómicos (T2); ou nível das operações (T3) que compõem o evento, a condição e a ação.

- ***Escalonamento da Transação/Aplicação:*** Depende da unidade de execução estabelecida pela granularidade do gatilho e pela granularidade da aplicação.

O uso de gatilhos ou regras para notificação, comunicação, e otimização é suportado por todos os modelos de execução. No entanto, uma granularidade de aplicação mais fina leva a maiores possibilidades no uso de regras. Por exemplo, se a granularidade da ativação é no nível de comando (A3), então podem-se definir regras para a interação com

o usuário, o que seria impossível com a granularidade de ativação a nível de primitivas (A4).

No caso da manutenção de integridade, é necessário que a integridade seja verificada (ou até corrigida) antes do commit da transação. Para isto é necessário ter o controle a nível de comando (A3), para que as regras possam ser executadas imediatamente antes do commit.

4.1. ESCALONAMENTO DA ATIVAÇÃO DAS REGRAS

Múltiplos gatilhos ou regras podem estar habilitados ao mesmo tempo. Esta situação, conhecida como conjunto *prontos para executar*, pode levar o sistema a um comportamento inconsistente. Vários algoritmos são propostos para resolver este problema, a saber:

- ***Execução em paralelo:*** Todos os gatilhos ou regras são executados em paralelo. Isto só pode ser feito se as regras não interferem entre si.
- ***Ordem seqüencial de execução:*** Todos os gatilhos ou regras são executados seqüencialmente, a ordem podendo ser tanto aleatória, por prioridade, ou outras.
- ***Execução simples de uma regra:*** Só uma regra é escolhida, com as mesmas opções que no caso anterior.

A execução de uma regra pode também ativar outras regras, o que é conhecido como ativação em cascata. Uma vantagem é o tratamento uniforme de todas as transações, mas a desvantagem é que possivelmente pode levar a situações de "livelock".

5. OTIMIZAÇÃO

Existem várias técnicas de otimização para execução de regras que variam segundo os diferentes objetivos dos sistemas ativos e das arquiteturas subjacentes. As características principais destas estratégias são classificadas em três grupos: otimização na execução das consultas, redução na execução de consultas, e otimização no armazenamento.

As estratégias do primeiro grupo são as mesmas que as otimizações de ordenação de operações usadas nos sistemas de bancos de dados convencionais e são descritas com detalhe em [Ull82].

No segundo grupo estão os algoritmos para a redução da quantidade de cálculo. Neste caso o objetivo é tentar excluir código redundante de subexpressões comuns e evitar execução de regras ou gatilhos que não são necessários. Há duas estratégias relevantes dentro deste grupo. A primeira é a otimização da avaliação da condição, como, por exemplo, avaliação incremental das condições. A segunda é a redução do tempo de computação das ações. Isto pode ser feito executando as regras o mais cedo possível, ou deixando-as para depois, quando talvez seja possível otimizar a execução através de abandono de regras supérfluas ou avaliação em conjunto [VK93].

O último grupo consiste em estratégias para a otimização no armazenamento de dados e regras. A indexação dos dados neste caso é feita tendo em consideração os predicados das regras, reduzindo assim a quantidade de objetos considerados no momento da avaliação da condição.

5.1. DETECÇÃO DE EVENTOS

É obvio que a implementação de um detector de eventos eficiente é crucial para ter um bom desempenho num BDA. Existem várias formas de detectar eventos, alguma das quais são descritas a seguir:

- ***Centralizada.*** Quando um evento é gerado, todas as regras são verificadas. É o mais fácil de implementar, mas também o de pior desempenho.
- ***Índices.*** Regras podem ser indexadas segundo os eventos que as ativaram. Assim, quando o evento ocorre, as regras que podem ser disparadas são encontradas rapidamente.
- ***Subscrição.*** Associação de regras a objetos geradores de eventos. Quando um evento é gerado, este é enviado a todas as regras que subscreveram aquele evento. No Sentinel é usado este mecanismo, mas existe um detector de eventos local à cada regra.
- ***Rede de Eventos.*** Para cada evento composto é construída uma rede. O sistema trabalha com uma combinação de redes que inclui todas as redes de todos os eventos compostos. Um evento pode participar em mais de uma composição, enquanto na combinação das redes só existe um estado para cada evento. A vantagem do uso destas regras é que os eventos compostos podem ser detectados passo a passo, dada a ocorrência de um evento primitivo, e não é preciso inspecionar um grande número de eventos primários armazenados no registro de eventos. Este mecanismo de detecção é implementado em alguns dos protótipos (que serão detalhados a seguir) usando autômatos finitos [GJS92b], redes de Petri [GD94] e grafos [CBB+89, Ber94].

6. ALGUNS EXEMPLOS DE SISTEMAS ATIVOS

O gerenciamento de regras e/ou gatilhos pode ser implementado separadamente do sistema de banco de dados, constituindo uma camada adicional que não está integrada ao núcleo do SGBD [BL92, SKM92]. Outros sistemas de bancos de dados ativos, tanto relacionais como orientado a objetos, foram projetados ou estendidos para tal fim. Ariel [Han89], HiPAC [CBB$^+$89], Ode [GJ92], Postgres [SJGP90], Samos [GD92], Starburst [Wid92] e Sentinel [CHS92] são os projetos mais destacados na área. A seguir são descritas as características ativas destes projetos.

6.1. ARIEL [HAN89]

Ariel é um sistema de gerenciamento de banco de dados implementado sobre Exodus [CFM+86] com um sistema de regras integrado. As principais questões de projeto são a integração do sistema de regras com o processamento de transações e a eficiência do sistema todo. O resultado é um sistema convencional de gerenciamento de banco de dados relacional estendido com um sistema de regras [Han89].

6.1.1. Regras

Ariel está baseado no modelo relacional. Usa um subconjunto da linguagem de consulta POSTQUEL [SHH88]. A linguagem de especificação de regras de Ariel é uma linguagem projetada para ambientes de banco de dados. A condição de uma regra pode basear-se numa combinação de eventos do banco de dados e/ou em um casamento sobre sequências de eventos.

A sintaxe das regras é a seguinte:

[**priority** *prioridade*]
[**on** *evento*]
[**if** *condição*]
[**then** *ação*]

- *prioridade*: fornece um controle sobre a ordem de execução das regras.
- *evento*: permite a especificação do evento que vai ativar a regra. Existem dois tipos de eventos: eventos temporais e eventos próprios dos bancos de dados, tais como append, delete, replace e retrieve.
- *condição*: pode se basear numa combinação de eventos de bancos de dados ou casamento de padrões sobre uma seqüência de eventos. As condições também podem testar condições de transição (baseado no estado prévio e atual).
- *ação*: especifica as ações a serem executadas quando a regra é disparada. A ação é uma seqüência de comandos do banco de dados.

6.1.2. Modelo de execução

As regras podem ser definidas para reagir a simples comandos do banco de dados, tal como delete ou replace. Portanto os eventos são detectados a um nível de comando, ou seja, granularidade de ativação A3. A ação é executada no fim do comando de mais alto nível onde aconteceu o evento. Ariel define um comando composto como um bloco ou lista de comandos. Um comando é chamado de *top-level* se não está aninhado dentro de um comando composto.

O tipo de granularidade de regra é T2, dado que o evento e a avaliação da condição estão separados da execução da ação. A ação é executada uma vez para cada um dos objetos no conjunto *prontos para executar,* e a execução da regra é parte da transação

onde a própria regra foi ativada, portanto se a ação da regra falhar, a transação é abortada.

A seleção da regra a ser executada está baseada em prioridades e tempo de ativação. As regras podem acessar tanto o estado do banco de dados, como o conjunto de variáveis do evento que causou sua execução.

6.1.3. Otimização

A avaliação do evento e a condição estão baseadas no algoritmo de Treat [HC+90], chamado A-Treat (Ariel-TREAT). Este algoritmo mantém incrementalmente conjuntos de objetos que satisfazem as condições. No caso em que um evento ativa uma regra, todos os objetos que satisfazem a condição da regra podem ser achados no conjunto.

6.2. HiPAC [CBB+89]

O nome HiPAC vem de ***Hi****gh* ***P****erformance* ***AC****tive database system*. Este projeto considera dois problemas no gerenciamento de restrições de dados: a manipulação de restrições temporais em banco de dados, e evitar o desperdício do polling das aplicações que usam regras.

Este projeto trabalha sobre um modelo de dados estendido próprio [DBB+88], e inclui construtores para representar suas regras. Estes construtores utilizam o mecanismo de regras ECA, conseguindo transformar o HiPAC num sistema de banco de dados orientado a objetos ativo.

6.2.1. Regras

No HiPAC as regras são tratadas como objetos. Existe uma classe de objetos do tipo regra e toda regra é uma ocorrência desta classe. O tratamento de regras como objetos permite que sejam relacionadas com outros objetos e possuam atributos. Além disso, podem ser criadas, modificadas ou excluídas da mesma forma que outros objetos [DBM88].

A sintaxe das regras é a seguinte:

identificador-da_regra
event *evento*
condition:
    coupling: *modo*
    query: *consulta*
action:
    coupling: *modo*
    operation: *operação*
[**timing-constraints** *lista-de-restrições-temporais*]
[**contingency-plans** *exeções*]

- *evento*: O evento é uma entidade que possui um identificador e uma lista de argumentos formais tipados. Os eventos podem ser: operações sobre o

banco de dados; temporais (por exemplo, o evento ocorre sempre às cinco da tarde); abstratos (eventos gerados pelas aplicações); compostos (por exemplo, é preciso que ocorram dois eventos em seqüência).

- *condição***:** A condição de uma regra é um objeto. A condição está baseada na lógica de Horn. Sua estrutura é descrita por duas funções: modo de acoplamento e uma coleção de consultas.

  O *modo* de acoplamento do objeto regra descreve a ligação entre a ocorrência do evento, a avaliação da condição e a execução da ação. Existem quatro possibilidades:

  - Imediato: a condição é avaliada quando o evento ocorre.
  - Diferido: a condição é testada no fim da transação.
  - Acoplado-dependente: a condição é testada numa transação separada, após o fim da transação que gerou o evento. Se a transação geradora aborta, então a condição não é testada.
  - Acoplado-independente: a condição é avaliada em uma transação separada, após o fim da transação que gerou o evento. A condição é sempre testada.

  As consultas que formam uma regra devem ser todas satisfeitas para que a condição seja verdadeira.

- *ação:* É um objeto complexo. Sua estrutura é definida por duas funções: modo de acoplamento (similar à definição da condição) e uma operação (que pode ser por exemplo uma mensagem para algum processo ou algum programa escrito na linguagem de manipulação de dados do sistema).

- *lista-de-restrições-temporais*: Corresponde a *deadlines*, prioridades, urgências ou funções de valores. Não são propriedades exclusivas de regras, mas podem ser acopladas a qualquer tarefa no sistema HiPAC.

- *exceções*: São ações alternativas que podem ser invocadas sempre que a ação especificada numa regra não possa ser executada.

As operações sobre os objetos regras são: *create* (cria uma regra), *delete* (elimina uma regra), *enable* (ativa uma regra para que possa ser usada), *disable* (desativa uma regra) e *fire* (faz com que a regra seja executada).

6.2.2. Modelo de execução

Um dos objetivos no projeto HiPAC é fornecer um bom tempo de resposta aos eventos críticos. É assim importante avaliar a condição logo após que aconteça o evento, e executar a seguir a ação. Para cumprir estes objetivos, as regras foram estendidas com os tipos de modo de acoplamento descritos.

A granularidade da regra é determinada pelo modo de acoplamento e, pode ser tanto T1, como T2 ou T3 [HLM88].

Se um evento ativa mais de uma regra, então para cada par condição-ação é criada uma sub-transação. Uma ação é executada uma vez para cada um dos objetos no *conjunto prontos para executar*. Estes objetos são passados para a ação por meio dos argumentos. As variáveis livres são ligadas ao conjunto de objetos que satisfaçam ao evento e à condição, sendo a ação executada uma vez para o conjunto.

6.2.3. Otimização

Como um dos objetivos de HiPAC é o desempenho, foram identificadas várias técnicas para a avaliação de condições complexas. Algumas são: otimização de condições múltiplas, gerenciamento de dados derivados, e avaliação incremental da condição. É usada uma estrutura de rede tipo Rete [For82] para o processamento das condições (indexação de dados).

Para otimizar a detecção de eventos é utilizado o mecanismo de rede utilizando um grafo orientado acíclico, chamado *signal graph*.

6.3. ODE [GJ91]

O banco de dados orientado a objetos Ode suporta o paradigma de objetos de C++. A interface com o usuário é a linguagem O++, que estende C++ fornecendo facilidades para a criação de objetos persistentes.

6.3.1. Restrições e Gatilhos

Ode fornece dois tipos de facilidades de regras: *restrições* para a manutenção da integridade do banco de dados, e *gatilhos* para a execução automática de ações dependendo do estado do banco de dados.

As restrições são usadas para manter a noção de consistência, o que é geralmente expresso com o sistema de tipos. Estas restrições, que são condições booleanas, são associadas às definições das classes, como é mostrado no exemplo seguinte. As restrições podem ser herdadas como qualquer outra propriedade dos objetos. Se uma restrição é violada e não é corrigida, a transação é abortada.

```
class exemplo{
...
public:
...
[soft] constraints:
        condição : ação
}
```

O teste da condição pode ser feito logo depois de acessar o objeto (conhecido como hard), ou diferido (chamado soft). Este último tipo permite violações temporais de restrições que são corrigidas com ações antes que o commit da transação seja executado.

Os gatilhos, como as restrições, monitoram o banco de dados verificando algumas condições, exceto que estas ultimas não representam violações de consistência. Um

gatilho é especificado na definição da classe e consiste de duas partes: o predicado do evento e a ação. Os gatilhos só podem ser aplicáveis àqueles objetos especificados.

trigger:
[**perpetual**] *nome-do-trigger* **(***params***) :**
[**within** *expressão* **?**] *condição* => *ação* [**:** *ação-do-time-out*]

Os gatilhos podem ser *once-only* (desativados logo após a primeira execução, usados por default) ou *perpetual* (reativados automaticamente depois de cada execução). Gatilhos podem ter parâmetros, e podem também ser ativados várias vezes com distintos valores como parâmetros. A *ação* associada a um gatilho é executada quando a *condição* (e *expressão*) é verdadeira, sempre e quando o gatilho esteja ativado. Após o time-out especificado, a *ação-do-time-out* é executada.

6.3.2. Modelo de Execução

Os gatilhos são associados a objetos, e são ativados explicitamente depois que o objeto foi criado. A ação é executada numa sub-transação separada, dependendo do commit da transação original. A palavra chave *independent* faz com que a execução da sub-transação seja executada de forma independente. Outros modos de acoplamento podem ser imediato, ou no fim da transação.

6.4. POSTGRES [SJGP90]

É uma extensão de um sistema de gerenciamento de bancos de dados relacional que inclui um sistema de gatilhos de propósito geral. O sistema de gatilhos pretende ser usado como uma linguagem para implementação de visões, visões materializadas, visões parciais, e procedimentos. O projeto fundamental do sistema de regras do Postgres é descrito em [SR86, SHH88, SJGP90, SHP89].

6.4.1. Regras

A primeira versão do Postgres usa um paradigma de ativação de regras, onde cada comando Postquel pode ter associado os modificadores: *always*, *refuse* ou *one-time*. A segunda versão do sistema de regras usa um enfoque de sistemas de produção mais tradicional.

define rule *nome-da-regra* [**as exception to** *nome-da-regra*]
on *evento* **to** *objeto* [[**from** *clausula*] **where** *clausula*]
then do [**instead**] *ação*

O *evento* é um dos seguintes: retrieve, replace, delete, append, new e old. New é usado com replace ou append, e old quando usa-se delete ou replace. A clausula *where* é igual às condições das consultas padrão dos sistemas de banco de dados (fórmulas em lógica de Horn). A *ação* é uma coleção de comandos Postquel mais as variáveis predefinidas *new* e *current*. Estas variáveis contêm os valores do objeto a modificar.

6.4.2. Modelo de Execução

O modelo de execução do sistema de regras de Postgres é definido a nível de tupla. Quando uma tupla é acessada, atualizada, inserida ou eliminada, existe uma tupla atual (current) (para os casos de retrieve, replace e delete), e uma tupla nova (new) (no caso de replace e append). Se o evento e a condição especificada são verdadeiras para a tupla atual, então a ação é executada. A granularidade da aplicação é a nível de operação (A4) e a granularidade a nível de regra é completa (T1). O cascateamento de regras não é descrito.

6.4.3. Otimização

São usadas duas diferentes estratégias de implementação para a otimização da execução das regras: execução a nível de tupla e re-escrita da consulta. A implementação a nível de tupla mantém trancas de eventos (event locks) nas tuplas. Estas trancas são colocadas em campos apropriados em todas as tuplas envolvidas na condição e segundo a especificação do evento da regra. Quando um evento acontece num campo marcado com um "event lock", a condição é testada. Se for verdadeira, a ação é executada. No processamento da consulta, pode-se saber logo quais são as regras a serem executadas.

A re-escrita é feita entre as etapas de parsing e otimização. A componente que re-escreve a consulta compila os comandos junto com as regras que estão ligadas aos comandos, evitando assim buscas dinâmicas de regras.

Dependendo da regra, uma das implementações de otimização é escolhida; ver [SJGP90] para maiores detalhes.

6.5. SAMOS [GD92]

Fornece as mesmas propriedades que todos os gerenciadores de banco de dados orientados a objetos (SGBDOO) como herança, tipos e operações definíveis pelo usuário, encapsulamento, dentre outros. O protótipo foi implementado baseado no SGBDOO comercial ObjectStore. Samos fornece, como Snoop [CM91] e ODE [GJ92], uma linguagem de eventos que inclui vários construtores para a especificação de eventos. Os eventos podem ser divididos em duas categorias: eventos primitivos (eventos de métodos, de valor, temporais ou abstratos) ou compostos (usando disjunção, conjunção, seqüência, vezes ou negação).

6.5.1. Regras

As regras são representadas como objetos, o que dá uma flexibilidade maior, podendo ser classificadas segundo seu relacionamento com as classes (ou objetos). Existem dois tipos de relações entre as regras e as classes. Regras *internas* à classe, que permitem operar ou acessar os valores dos objetos, ou *externas*. As internas formam parte da definição da classe, e estão encapsuladas dentro das instâncias. Condições e ações das regras internas podem operar diretamente com os valores. As regras externas à classe podem ser definidas por qualquer usuário ou aplicação independente da definição da

classe. Na definição da classe especifica-se quando precisa ser avaliada a condição, ou quando a ação deve ser executada (modo de acoplamento) imediato (logo após o evento), diferido (antes do commit), ou independente (numa transação independente).

6.5.2. Modelo de Execução

A avaliação da condição e a execução da ação são implementadas como sub-transações. A ordem de execução das regras disparadas ao mesmo tempo usa um mecanismo de prioridades.

O projeto Samos estende a arquitetura de um sistema de banco de dados (passivo) orientado a objetos (ObjectStore) com novas componentes, a saber: *analisador*, *gerenciador de regras e eventos*, *detector de eventos*, e *executor das ações*. O *analisador* é responsável pela passagem das regras e eventos ao gerenciador correspondente (estes manipulam a base de regras e a história dos eventos, respectivamente). O *detector de eventos* precisa manter a estrutura de dados necessária para a detecção dos eventos. Logo após o evento ser detectado, é inserido no registro de eventos. Baseado nesse registro, o *gerenciador de regras* determina quais regras precisam ser executadas, e o *executor* é encarregado de avaliar as condições e executar a ações.

6.5.3. Otimização

A implementação de Samos esta construída sobre ObjectStore como uma caixa preta, havendo portanto uma perda de desempenho.

É obvio que a implementação de um detector de eventos eficiente é crucial para ter uma boa performance num sistema de bancos de dados ativos. As Redes de Petri foram usadas para modelagem e detecção de eventos. Para cada construtor há um padrão de Rede de Petri. O sistema trabalha com uma combinação de redes de Petri que inclui todas as redes de todos os eventos compostos. Um evento pode participar em mais de uma composição, enquanto na combinação das Redes só existe um estado para cada evento. A vantagem do uso destas regras é que os eventos compostos podem ser detectados passo a passo, dada a ocorrência de um evento primitivo, e não é preciso inspecionar um grande número de eventos primários armazenados no registro de eventos [GD94].

6.6. STARBURST [WF90]

O Starburst é um sistema de gerenciamento de banco de dados relacional estendido. Há duas formas de extensão, conhecidos como métodos de armazenamento e *attachments*. Os métodos de armazenamento provêem métodos alternativos para implementação de tabelas. Os attachments provêem caminhos de acesso, restrições de integridade e extensões de gatilhos.

6.6.1. Regras

A sintaxe da linguagem para expressar regras em Starburst é descrita a seguir:

create rule *nome* **on** *tabela*
when *operações*
[**if** *condição*]
then *ação*
[**precedes** *lista-de-regras*] [**follows** *lista-de-regras*]

As *operações* são uma ou mais dentre inserted, deleted, ou updated(c_1,...,c_n), onde c_1,...,c_n são atributos de relações. A *condição* é um predicado SQL sobre o banco de dados. A ação é uma seqüência de operações do banco de dados, incluindo comandos de manipulação SQL, comandos de definição, e o comando rollback. Os comandos podem ser também alter, drop, deactivate, e activate sobre as regras. As seções *precedes* e *follows* são usadas para ordenar parcialmente o conjunto de regras [Wid92].

6.6.2. Mecanismo de Execução

As regras são processadas logo após a finalização das transações, sendo também possível adicionar outros pontos de processamento dentro das transações.

A semântica está baseada em transições, que são mudanças de estado do banco de dados que resultam da execução de comandos SQL. A execução da transação é a primeira transição relevante, e algumas regras podem ser disparadas por essa transação. Quando as ações de uma regra são executadas, outras transições são criadas. Estas podem disparar regras adicionais, ou as mesmas regras várias vezes. O processamento das regras é um algoritmo interativo com as seguintes passos:

1. Uma regra R que foi disparada é selecionada para examinar a sua prioridade com respeito a outras regras nas mesmas condições.
2. A condição da regra R é avaliada.
3. Se a condição de R é verdadeira, a ação correspondente é executada.

O processamento termina quando a operação rollback é executada, ou quando não há mais regras para disparar.

As condições e ações fazem referência ao estado atual do banco de dados, através de operações de seleção de SQL. Além disso, as condições e ações das regras fazem referência a tabelas de transição, que são tabelas lógicas que refletem as mudanças que aconteceram durante a transição do disparo das regras.

O sistema de regras inclui mecanismos de controle de concorrência que asseguram a consistência com respeito aos dados e regras, e na ordenação das regras. Também inclui mecanismos para a recuperação ante rollbacks das estruturas de dados do próprio sistema de regras.

6.7. SENTINEL [CAM93]

Este projeto integra as regras de tipo ECA, introduzidas em HiPAC [CBB+89], a um sistema de banco de dados orientado a objeto, estendendo também a linguagem de especificação de eventos. Os objetivos são: usar o sistema resultante para monitoramento

do próprio banco de dados, fornecer suporte cooperativo na resolução de problemas, e suportar SGBDs ativos multimídia para aplicações científicas [CHS92].

6.7.1. Regras

Os objetos C++ convencionais foram estendidos com uma *interface de eventos*. Com isto, é possível gerar eventos quando os métodos são invocados, e propaga-los a outros objetos. Os eventos podem ser gerados antes ou depois da execução do método. As classes *reativas* são aquelas que, além da definição tradicional, têm a especificação da interface de eventos. Os eventos são definidos pelo usuário na própria definição da classe.

A especificação de eventos está baseada na linguagem Snoop [CM91]. Esta linguagem propõe uma hierarquia de eventos constituída por eventos primários (primitivos) e compostos. São definidos operadores para eventos, como seqüência, disjunção, etc. A noção de parâmetros é usada para computar os parâmetros dos eventos complexos. A especificação de eventos e regras pode ser feita em qualquer uma das classes. São suportados os modos de acoplamento imediato ou diferido.

As regras, que são do tipo ECA, são criadas, modificadas e eliminadas da mesma forma que os outros objetos, tendo assim um tratamento uniforme. As regras são também entidades separadas que existem independentemente de outros objetos no sistema. Cada regra tem: uma identificação (para poder associa-la a outros objetos), uma associação a um objeto de tipo evento, e dois métodos públicos: condição e ação. As regras são ser associadas a outros objetos utilizando o mecanismo de subscrição [CAM93], que é descrito a seguir.

6.7.2. Modelo de Execução

O sistema Sentinel é desenvolvido usando o gerenciador de banco de dados orientado a objetos Zeitgest. Para incorporar as regras no sistema foram incluídas as classes: Reativa, Notificável, Evento, Regra. A hierarquia tem a classe persistente *zg-pos* (predefinida no sistema), como super classe de Reativa e Notificável, e as classes Evento e Regra são subclasses desta última. Assim, Eventos e Regras podem receber eventos propagados pelos objetos reativos. Para associar Regras a objetos é introduzido o mecanismo de *subscrição*. Isto permite aos objetos notificáveis (neste caso as Regras) subscrever dinamicamente aos eventos gerados por objetos reativos.

6.7.3. Otimização

O esquema da subscrição tem a vantagem que uma mesma regra pode ser aplicada a objetos de tipos diferentes, o que é mais eficiente que definir a mesma regra múltiplas vezes para cada tipo de objeto. Outra vantagem deste esquema é que as regras que estão subscritas a um objeto reativo só são testadas quando este objeto gera eventos. Isto tem um ganho em desempenho em relação ao esquema centralizado, onde todas as regras definidas no sistema são testadas quando um evento é gerado.

7. CONCLUSÕES

Foram discutidas as principais características dos bancos de dados ativos utilizando como marco para sua descrição as seguintes categorias: definição das regras, o modelo de execução e a otimização da execução.

Também foram descritos, utilizando este marco, os principais protótipos que foram desenvolvidos na área. Na tabela 1 são resumidas características destes protótipos.

8. REFERÊNCIAS

[ABD+92] Atkinson, Bancilhon, DeWitt, Dittrich, Maier and Zdonik: The Object-Oriented Database System Manifesto. Building an Object-Oriented Database System, F. Bancilhon, C. Delobel and P. Kandellakis (editors), Morgan Kaufmann, 1992.

[Ber94] Berndtsson, M.: Reactive Object-Oriented Databases and CIM. In Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA '94), pages 769-728, Athens, Greece, September 1994.

[BL92] Berndtsson, M; Lings, B.: On Developing Reactive Object-Oriented Databases. IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 15, No. 1-4, pages 31-34, December 1992.

[Buch94] Buchmann, A.: Active Databases. Tutorial. IX School of Computing, Recife, Brazil, July 1994.

[CAM93] Chakravarthy, S.; Anwar, E. and Maugis, L.: Design and Implementation of Active Capability for an Object-Oriented Database. Technical Report UF-CIS TR-93-1, CIS Department, University of Florida, September 1993.

[CBB+89] Chakravarthy, S.; Blaustein, B.; Buchman, A.: Carey, M.; et. al.: HiPAC: A Research Project in Active, Time-Constrained Database Management. Final Technical Report XAIT-89-02 (187), July 1989.

[CFM+86] Carey, M.J.; Frank, D; Muralkrisna, M., DeWitt, D.J.; Graefe, G.; Richardson, J.E.; Shekita, E.J.: The Architecture of the EXODUS Extensible DBMS. Readings in Database Systems, M. Stonebraker (editor), pages 488-502, Morgarn-Kaufmann Publishers, San Mateo, California, 1988.

	Ariel	HiPAC	Ode	Postgres	Samos	Starburst	Sentinel
Modelo	Relacional estendido (Exodus)	Orientado a Objeto	O++ (C++ persist.)	Relacional estendido	Orientado a Objeto (Object-Store)	Relacional estendido	Orientado a Objeto (Zeitgest)
Especificação das Regras E-C-A							
Condição	Subconj. de Postquel	Subconj. das consultas	Expressão C++	Postquel	Linguagem de consulta	SQL	C++ / Zeitgest
Evento	append/ delete/ replace/ temporais	BD / relógio / notificações externas	update	retrieve/ replace/ delete/ append/ new / old	BD / temporais/ métodos / abstratos/ transações	insert / update / delete	BD / transações / temporais / métodos
Ação	Subconj. de Postquel	DML/ Procedim. Externos	O++	Postquel	DML	DML	C++ / Zeitgest
Regras							
Herança	Não	-	Sim	Não	Sim	Não	Sim
Eventos							
Mecanismo de Detecção	baseado em TREAT [HC+90]	*signal graphs*	autômato de estado finitos	Índices	Redes de Petri	Índices (*Rule catalogs*)	Subscrição(Detecção local à regra)
Composição	Não	Sim	Sim	Não	Sim	Não	Sim
Condição							
Otimização	Re-escrita da consulta	Avaliação incremental / Sub-expressões comuns / Rete [For82]	duas avaliações	re-escrita consulta / a nível de tupla	Depende de Object-Store	-	Depende de Zeitgest
Modelo de Execução							
Resolução de conflitos	prioridade	concorrência	- (concorr.)	-	prioridade	prioridade	concorrência
Modos Acoplamento	Imediato	Imediato/ Diferido/ Separado (dependen /indepen.)	Imediato / Diferido	Imediato	Imediato / Diferido-independ.	Diferido	Imediato / Diferido

Tabela 1. Resumo das características dos principais protótipos.

[CHS92] Chakravarthy, S.; Hanson, E.; Su, S.Y.W.: Active Database/Knowledge Base Research at the University of Florida. IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 15, No. 1-4, pages 35-39, December 1992.

[CM91] Chakravarthy, S.; Mishra, D.: An Event Specification Language (Snoop) for Active Databases and its Detection. Technical Report UF-CIS TR-91-23, CIS Department, University of Florida, September 1991.

[CM93] Chakravarthy, S. and Mishra, D.: Snoop: an Expressive Event Specification Language for Active Databases. Technical Report UF-CIS-TR-93-007, University of Florida, March 1993.

[CN90] Chakravarthy, S.; Nesson, S.: Making an Object-Oriented DBMS Active: Design, Implementation and Evaluation of a Prototype. In Proceedings of the International Conference on Extending Database Technology, Venice, March 1990.

[CW90] Ceri, S.; Widom, J.: Deriving Production Rules for Constraint Maintenance. In Proceedings of 16th International Conference on Very Large Data Bases (VLDB '90), pages 566-577, Brisbane, Australia, August 1990.

[DBB+88] Dayal, U.; Blaustein, B.; Buchmann, A.; Chakravarthy, S. et al.: The HiPAC Project: Combining Active Databases and Timing Constraints. ACM SIGMOD Record, Vol. 17, No. 1, pages 51-70, March 1988.

[DBM88] Dayal, U.; Buchmann, A.; McCarthy, D.: Rules are objects too: a knowledge model for an active, object-oriented database system. In Proceedings of the 2nd International Workshop on Object-Oriented Database Systems, Lecture Notes in Computer Science 334, Springer 1988.

[For82] Forgy, C.L.: Rete A Fast Algorithm for the many Pattern/many Object Pattern Match Problem. Artificial Intelligence, Vol. 19, pages 17-37, 1982.

[GD92] Gatziu, S; Dittrich, K.R.: SAMOS: an Active Object-Oriented Database System. IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 15, No. 1-4, pages 23-26, December 1992.

[GD93] Gatziu, S. and Dittrich, K.: Events in an Active Object Oriented Database System. In Proceedings of the International Workshop on Rules in Database Systems, pages 23-29, August 1993.

[GD94] Gatziu, S. and Dittrich, K.: Detecting Composite Events in Active Database Systems Using Petri Nets. In Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE '94), Huston, Texas, February 1994.

[GJ91] Gehani, N.; Jagadish, H.V.: Ode as an Active Database: Constraints and Triggers. in Proceedings of the 17th International Conference on Very Large Data Bases (VLDB '91), pages 327-336, 1991.

[GJ92] Gehani, N.; Jagadish, H.V.: Active Database Facilities in Ode. IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 15, No. 1-4, pages 19-22, December 1992.

[GJS92a] Gehani, N.; Jagadish, H. and Shmueli, O.: Compose: a System for Composite Event Specification and Detection. Technical report AT&T Bell Laboratories, December 1992.

[GJS92b] Gehani, N.; Jagadish, H. and Shmueli, O.: Event Specification in an active object oriented database. In Proceedings of the International Conference of Maganement of Data (SIGMOD '92), pages 81-90, 1992.

[Han89] Hanson, E.N.: An Initial Report on the Design of Ariel: A DBMS With an Integrated Production Rule System. ACM SIGMOD Record, Vol. 18, No. 3, pages 12-19, September 1989.

[HC+90] Hanson, E.; Chaabouni, M.; Kim, C. and Wang,Y.: A Predicate Matching Algorithm for Database Rule Systems. In Proceedings of the International Conference on Management of Data (SIGMOD '90), May 1990.

[HLM88] Hsu, M; Ladin, R.; McCarthy, D.R.: An Execution Model for Active Data Base Management Systems. In Proceedings of the 3rd International Conference on Data and Knowledge Bases, Jerusalem, June 1988.

[LT92] Ling, T.W.; Teo, P.K.: On Rules and Integrity Constraints in Database Systems. Information and Software Technology, Vol. 34, No. 3, March 1992.

[MA92] Medeiros, C.B.; Andrade, M. J.: Implementing Integrity Control in Active Databases. Relatório Técnico DCC, UNICAMP, July 1992.

[Mor84] Morgestein, M.: Constraint Equations: Declarative Expression of Constraints with Automatics Enforcement. In Proceedings of the Tenth International Conference on Very Large Data Base (VLDB '84), pages 291-300, Singapore, August 1984.

[SHH88] Stonebraker, M.; Hanson, E.; Hong, C.H.: The Design of the Postgres Rule System. Readings in Database Systems, M. Stonebraker (ed.), Morgan-Kaufmann, 1988.

[SHP89] Stonebraker, M.; Hearst, M.; Potamianos, S.: A Commentary on the Postgres Rule Manager. SIGMOD Record, Vol. 18, No. 3, September 1989.

[SJGP90] Stonebraker, M.; Jhingran, A.; Goh, J.; Potamianos, S.: On Rules, Procedures, Caching and Views in Data Base Systems. In Proceedings of the International Conference on Management of Data (SIGMOD '90), pages 281-290, Atlantic City, May 1990.

[SR86] Stonebraker, M.; Rowe, L.: The Design of POSTGRES. In Proceedings of the International Conference on Management of Data (SIGMOD '86), Washington, D.C., June 1986.

[SR88] Sellis, T.; Raschid, L.: Implementing Large Production Rules System in a DBMS Enviroment: Concepts and Algorithms. In Proceedings of the International Conference on Management of Data (SIGMOD '88), pages 281-290, Chicago, June 88.

[Ull82] Ullman, J.D.: Principles of Database Systems. Computer Science Press, 1982.

[VK93] Van der Voort, M.H.; Kersten, M.L.: Facets of Database Triggers. Internal Report of CWI, 1009 AB Amsterdam, Netherlands, April 1993.

[WF90] Widom, J. and Finkelstein, S.: Set Oriented Production Rules in Relational Database Systems. In Proceedings of the International Conference on Management of Data (SIGMOD '90), pages 259-270, May 1990.

[Wid92] Widom, J.: The Starburst Rule System: Language Design, Implementation, and Applications. IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 15, No. 1-4, pages 15-18, December 1992.

[Wid93] Widom, J.: Deductive and Active Databases: Two Paradigns or Ends of a Spectrum?. IBM Alamden Research Report, RJ9268 (81832), March 1993.