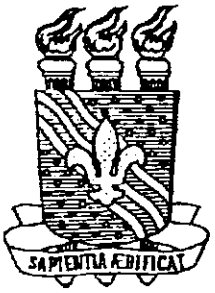



**Universidade Federal da Paraíba - UFPB**

**Centro de Ciências e Tecnologia - CCT**

**Departamento de Sistemas e Computação - DSC**

**Coordenação de Pós-Graduação em Informática – COPIN**

# Um Engenho Ativo para a Transformação de SGBDs Passivos ou Semi-Ativos em SGBDs Ativos

**Trícia Souto Santos**

Campina Grande
1998

**Trícia Souto Santos**

# Um Engenho Ativo para a Transformação de SGBDs Passivos ou Semi-Ativos em SGBDs Ativos

*Dissertação submetida ao Curso de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, em cumprimento às exigências parciais para obtenção do grau de Mestre em Informática.*

**Orientador:** Marcus Costa Sampaio, Dr.
**Linha de Pesquisa:** Sistemas de Informação e Bancos de Dados
**Área de Concentração:** Ciência da Computação

Campina Grande
Novembro de 1998

Ficha Catalográfica

Santos, Tricia Souto

S237E

Um Engenho Ativo para a Transformação de SGBDs Passivos ou Semi-Ativos em SGBDs Ativos – Campina Grande: CCT/COPIN da UFPB, Novembro de 1998, xxxp.

Dissertação (mestrado) – Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, campina grande, 1998.

Orientador: Marcus Costa Sampaio, Dr

1. Bancos de Dados Ativos
2. Regras Ativas
3. Sistemas de Gerência de Bancos de Dados Ativos

CDU – 681.3.07B

# UM ENGENHO ATIVO PARA A TRANSFORMAÇÃO DE SGBDs PASSIVOS OU SEMI-ATIVOS EM SGBDs ATIVOS

**TRÍCIA SOUTO SANTOS**

**DISSERTAÇÃO APROVADA EM 26.11.98**

**PROF. MARCUS COSTA SAMPAIO, Dr.**
**Presidente**

**PROF. ULRICH SCHIEL, Dr.**
**Examinador**

**PROF. ASTERIO KIYOSHI TANAKA, Ph.D**
**Examinador**

**CAMPINA GRANDE - PB**

A Luísa, minha filha, por ter se tornado o motivo principal da minha vida e a minha mãe, D. Graça, por ter sido a mãe da minha filha durante os momentos em que precisei estar ausente.

# RESUMO

Hoje a maioria dos Sistemas Gerenciadores de Bancos de Dados – SGBDs presentes no mercado possuem funcionalidades ativas, ou seja, respondem automaticamente a alguns eventos ocorridos no Banco de Dados. Porém, essas funcionalidades são, geralmente, limitadas. Neste trabalho, é proposta uma camada ativa, denominada de Engenho Ativo, a ser colocada no topo de SGBDs Relacionais comerciais, com o propósito de estender suas funcionalidades ativas sem que sejam necessárias alterações nos seus códigos-fonte.

# ABSTRACT

The most Database Manager Systems – DBMS available actually present active characteristics. It means they respond automatically to some database events. However these characteristics have, in general, constraints. In the present dissertation an active layer, named Active Engine, is proposed. It is to be inserted at the top of comercial relational DBMS, aiming to extend their active characteristics with no changes in their source codes.

# AGRADECIMENTOS

Agradeço a Deus, por ter me permitido viver e vencer mais esta etapa.

Agradeço as minhas irmãs: Isana e Eugra, aos meus avós: Zuzu e Lourdes, a Deda e aos meus primos: Alysson, Kalliop e Renaly, pelo carinho e principalmente ao meu pai, Euclides, que soube transmitir para as suas filhas a sabedoria da vida e que sempre me incentivou e trabalhou para que eu pudesse vencer.

Agradeço ao professor e orientador Dr. Marcus Costa Sampaio, por ter acreditado em mim, e ao pessoal da COPIN, pela paciência em resolver os meus problemas. Agradeço especialmente a Aninha, pelo incentivo durante todo o processo.

Agradeço aos amigos e colegas que estiveram, de longe ou de perto, torcendo por mim. Principalmente a Maurício Galimberti pela alegria que ele sempre me transmitiu.

Agradeço aos meus colegas do LMRS, principalmente a turma do setor de informática e especialmente a Iana e Ricardo Lima, pelo apoio e compreensão nas horas em que precisei trocar o trabalho pelo estudo.

Agradeço aos meus colegas Vicente Maia, pelo suporte via linha telefônica, e Marinaldo Nunes, pelas linhas e mais linhas de código.

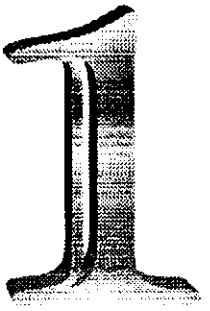
Agradeço a minha tia Xaxe pelo simples fato dela existir.

Finalmente, agradeço a todos que, direta ou indiretamente, contribuiram para a conclusão deste trabalho.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# SUMÁRIO



# Introdução

## 1.1 Motivação

Atualmente, raras são as organizações que não possuem um Sistema Gerenciador de Bancos de Dados - SGBD, para armazenar e gerenciar suas informações. Porém, a maioria destes sistemas não gerencia os aspectos dinâmicos do processamento da informação, ou seja, não têm a capacidade de responder automaticamente a eventos ocorridos no Banco de Dados - BD - e/ou nas aplicações. Estes sistemas pertencem à classe dos SGBDs passivos.

Em contraposição aos SGBDs passivos, surgiram os SGBDs ativos, chamados a seguir de SGBDAs. Os SGBDAs são sistemas de monitoramento automático das mudanças de estado do BD. Têm a capacidade de executar ações pré-definidas como reação automática a eventos, também pré-definidos e gerados pela manipulação de informações no BD [BP91][S93].

A modelagem dos aspectos dinâmicos tem feito uso de um poderoso formalismo denominado Regras ECA (Evento, Condição e Ação)[DBM88]. Na forma mais geral, uma regra ECA consiste de um agente disparador denominado **Evento** caracterizado pela ocorrência de algum fato no BD ou fora dele; de uma **Condição** que consiste de uma expressão booleana a ser avaliada e de uma **Ação**, compreendendo um conjunto de procedimentos pré-definidos. Funciona assim: quando um evento ocorre, uma regra correspondente a este evento é disparada, isto é, a condição é avaliada e se for satisfeita, a ação relativa é executada. Os detalhes do monitoramento de eventos e execução de regras são mais complexos que esta breve descrição, pois variam consideravelmente de sistema para sistema mas, no geral, este é o paradigma seguido por todos os SGBDAs [CW96].

As novas versões de SGBDs comerciais possuem algumas funcionalidades ativas. Porém, ainda não podem ser consideradas como SGBDs verdadeiramente ativos, por causa das muitas limitações em suas funcionalidades ativas. O Oracle Versão 7.3, por exemplo, implementa *triggers* (gatilhos), agentes que disparam um conjunto de procedimentos automaticamente, dada alguma alteração nos dados de uma tabela do BD. Uma das muitas limitações dos *triggers* é a restrição que eventos só podem ser definidos como a ocorrência dos comandos SQL de atualização de tabelas, `INSERT`, `DELETE` e `UPDATE` [D89]. Além disto, eventos no Oracle 7.3 são simples, ou seja, só pode existir um agente disparador.

Ao contrário de eventos simples no Oracle 7.3, a possibilidade de definição de eventos compostos é uma funcionalidade ativa bastante desejável em um SGBD [AMC93]. Entende-se por um evento composto uma expressão de eventos simples ou compostos conectados por operadores lógicos. Também, diferentemente do Oracle 7.3, onde os eventos são restritos a notificações no BD, eventos simples devem poder ser ocorrências no BD ou fora dele. Por exemplo, uma chamada a um procedimento definido pelo usuário também deve poder ser considerado um evento simples.

Para atender a um mercado cada vez mais exigente em matéria de SGBDAs e levando em consideração a base instalada existente, pode-se partir de um SGBD passivo ou fracamente ativo (ou ainda, semi-ativo) e implementar no seu topo um engenho ativo de maneira rápida e sem muito esforço de implementação, seguindo a idéia de

arquitetura em camadas [CW96], e disponibilizar assim um conjunto de novas funcionalidades ativas ao SGBD. Esta idéia do engenho ativo tem duas vantagens importantes:

a) um SGBD passivo ou semi-ativo pode ser transformado em um verdadeiro SGBDA sem grandes modificações no seu núcleo;
b) a interface com o engenho ativo e pelo menos boa parte de sua implementação pode ser comum a vários SGBDs diferentes (portabilidade do engenho).

Seguindo esta estratégia, surgiu a motivação para especificar e implementar um engenho ativo visando melhorar as funcionalidades ativas dos diversos SGBD comerciais presentes no mercado.

## 1.2 Objetivos

O objetivo dessa proposta é a especificação e implementação de um engenho ativo no topo de SGBDs comerciais visando aumentar a capacidade ativa dos mesmos. Como estudo de caso foi utilizado o Oracle Versão 7.3 para ambiente Windows NT 4.0.

A seguir, são dados exemplos de algumas das melhorias que foram proporcionadas ao Oracle Versão 7.3, no que diz respeito às suas funcionalidades ativas. Para efeito de comparação, o Oracle com as novas funcionalidades ativas é chamado de Oracle Ativo.

Eventos:

Oracle 7.3: Operações SQL `INSERT`, `DELETE` e `UPDATE` sobre tabelas do BD

Oracle Ativo: Definidos pelas aplicações

Ações:

Oracle 7.3: Conjunto de comandos PL/SQL[1].

Oracle Ativo: Chamadas a procedimentos executáveis codificados em qualquer linguagem.

[1] - Linguagem de desenvolvimento de aplicações do Oracle7.3

Como complemento importante do trabalho, é mostrado como o engenho ativo para o Oracle 7.3 pode ser facilmente adaptado para ser implementado no topo de outros SGBDs como Informix, Sybase, dentre outros.

## 1.3 Organização do Trabalho

Os capítulos da dissertação estão estruturados da seguinte forma:

O primeiro capítulo é esta introdução.

O segundo capítulo apresenta os principais conceitos de SGBDA, incluindo uma sintaxe e uma semântica de execução de regras. Algumas arquiteturas de transformação de SGBDs passivos em ativos são também apresentadas e analisadas. Por fim, é feita uma classificação de alguns SGBDs comerciais com funcionalidades ativas.

O terceiro capítulo traz a especificação de uma camada ativa a ser colocada no topo de qualquer SGBD relacional, a fim de incrementar as funcionalidades ativas já oferecidas. Este módulo é denominado de Engenho Ativo, e estende a especificação de *triggers* do padrão SQL3 [MS93].

No quarto capítulo são feitas considerações sobre o projeto do Engenho Ativo. A interface do Engenho Ativo é apresentada e são discutidos alguns detalhes dos algoritmos utilizados para a sua implementação.

No quinto capítulo são relatados os detalhes da implementação do Engenho Ativo sobre o SGBD Oracle Versão 7, bem como as experiências obtidas com a mesma.

No sexto capítulo são discutidas algumas aplicações do Engenho Ativo.

A dissertação se encerra com as conclusões e as propostas de trabalhos futuros (sétimo capítulo).

Os algoritmos e as estruturas de dados usadas na implementação do protótipo são matéria dos apêndices.

# Bancos de Dados Ativos - Um Breve Panorama

Neste capítulo são apresentados os conceitos de SGBDA e são analisadas as características ativas ideais que devem estar presentes em um SGBD para este ser considerado verdadeiramente ativo. É feita uma discussão sobre as arquiteturas de transformação de SGBDs passivos em ativos e, encerrando o capítulo, é apresentada uma síntese comparativa das características ativas dos principais SGBDs comerciais, dos padrões SQL92 e SQL3 e de uma arquitetura chamada *Alert*.

## 2.1 Bancos de Dados Ativos

### 2.1.1 Conceitos e Funcionalidades

Segundo [CW96], um SGBD, para ser considerado ativo, deve se preocupar com os aspectos dinâmicos da informação. Em outras palavras, os SGBDs não devem tratar apenas os dados presentes no BD num determinado instante e sim, devem se preocupar com as ocorrências que possam envolver os dados em um instante qualquer da execução do BD e modelar essas ocorrências a fim de que o banco se torne apto a reagir, automaticamente, a medida que estas acontecerem.

Mais precisamente, um SGBDA deve permitir definir e implementar Regras Ativas, também chamadas de Regras Evento-Condição-Ação, ou Regras ECA [DBM88]. Uma Regra ECA é constituída de um evento (E), uma condição (C) e uma ação (A). A maior parte dos exemplos dados a seguir são voltados para BDs Relacionais. O leitor será devidamente chamado à atenção nos casos de Regras ECA em outros tipos de BDs. Vale salientar que uma regra ativa *stricto sensu* é o par Condição-Ação e é disparada pela ocorrência de um determinado Evento (*trigger*, gatilho). A seguir, são explicados em detalhes os conceitos de Evento, Condição e Ação.

#### Evento

Eventos são ocorrências que causam a execução de alguma regra ativa *stricto sensu*. Eventos podem ser do tipo:

- operações sobre o BD: englobam operações de manipulação de dados como `SELECT`, `INSERT`, `DELETE` ou `UPDATE` da linguagem relacional SQL, por exemplo;
- temporais: podem ser absolutos (20 de junho de 1997, às 15:30), repetitivos (diariamente às 18:00) ou intervalos periódicos (de 30 em 30 minutos);
- definidos pela aplicação: a aplicação define um procedimento, o *login* de um usuário por exemplo, como um evento, e, a cada vez que este procedimento é executado, uma ou várias regras são disparadas.

Os eventos descritos até agora são considerados eventos simples. Pode-se definir ainda eventos compostos [JS94], eventos parametrizados e eventos lógicos.

Denomina-se evento composto a combinação de eventos simples e/ou outros eventos compostos, essa combinação podendo ser feita através de:

- operadores lógicos (*and*, *or*, *not*): sendo E1, E2 e E3 eventos simples, um evento composto E4 pode ser definido da seguinte forma: E4 = E1 and (E2 or E3). Ou seja, E4 acontecerá quando E1 acontecer e E2 ou E3 acontecer;
- uma sequência ordenada de eventos (simples ou compostos): Sendo E1 e E2 eventos, pode-se definir E3 como a sequência (E2,E1) onde E3 acontecerá quando E2 e E1 acontecerem nesta ordem;
- composição temporal: quando ocorre, na expressão de eventos, um evento temporal.

Nos eventos parametrizados, como é evidente, os eventos podem receber valores de variáveis-parâmetros. Esses valores podem ser referenciados na condição e na ação da regra. Por exemplo, nos BDs Orientados a Objeto, pode-se definir como evento a chamada a um método M, e passar como parâmetro o identificador do objeto que chamou o método M.

Um evento pode também ser definido como o efeito combinado de vários eventos. Por exemplo, suponha que um evento seja definido como a inserção de um valor qualquer V' no BD. Porém, se ocorrer um `INSERT`(V) seguido de um `UPDATE`(V=V'), o efeito produzido é o mesmo da ocorrência de um `INSERT`(V'). Casos assim caracterizam eventos lógicos, eventos que levam em consideração o resultado final obtido e não os passos efetuados para conseguir tal resultado.

**Condição**

A condição de uma regra ativa especifica uma expressão booleana que é avaliada

antes da ação ser executada. Na realidade, ela vai permitir ou não a execução da ação relacionada àquela regra. Condições podem ser definidas como:

- predicados na linguagem do BD: por exemplo, a condição é definida como um predicado da linguagem SQL (cláusula `WHERE`), com todo o poder de expressão desta linguagem [MS93];
- predicados restritos na linguagem do BD: idêntico ao anterior, porém com algumas restrições como, por exemplo, não permitir o uso de funções de agregação no predicado. Estas restrições são impostas geralmente por questões de desempenho;
- consultas ao BD: retornam os valores lógicos *FALSE* se a consulta não retornar valores, ou *TRUE*, caso contrário;
- chamadas a funções: essas funções podem ou não acessar o BD. Se a função retornar *TRUE* a condição é satisfeita, se retornar *FALSE* a condição não é satisfeita.

A linguagem de definição de condições deve permitir referências aos parâmetros da linguagem de definição de eventos, se estes existirem.

**Ação**

Quando ocorre um evento e a condição avaliada resulta em um valor verdadeiro, a **ação** (na realidade, um conjunto de comandos) relacionada àquela regra é executada. Um comando pode ser:

- modificação de dados do BD: por exemplo, a execução de operações SQL do tipo `INSERT`, `DELETE` e `UPDATE`;
- recuperação de dados do BD: por exemplo, a execução de operações SQL do tipo `SELECT`;
- outros comandos de BD: comandos de definição de tabelas, operações de controle de transações como `commit` e `rollback`, operações de segurança como `grant` e `revoke`, dentre outros comandos;
- chamada a procedimentos do usuário: qualquer comando de execução de um

procedimento do usuário, numa linguagem qualquer, deve poder fazer parte da ação de uma regra.

Um SGBDA deve oferecer também comandos para manipulação de regras, como por exemplo: *Create* (criação), *Delete* (exclusão), *Modify* (modificação) e *Activate* ou *Deactivate* (ativação/desativação) de regras.

### 2.1.2 Semântica de Execução de Regras

Denomina-se **semântica de execução de regras** o comportamento do BD em relação à linguagem de regras, bem como a interação do processamento das regras com o processamento normal das consultas e transações. A semântica de execução de regras pode se tornar muito complexa, devido ao grande número de alternativas de implementação.

A seguir, é apresentado um algoritmo, simples e iterativo, de execução de regras, mas que oferece várias maneiras de ser estendido ou modificado para obter um modelo de execução mais completo.

*Enquanto um evento ocorrer*
*encontre a regra R disparada pelo evento*
*avalie a condição de R*
*se a condição de R for verdadeira*
*então execute a ação de R*

Sucintamente, o algoritmo verifica repetidamente eventos, avalia sua condição e, se esta for satisfeita, executa a ação relacionada. Pode-se ver que o algoritmo processa uma regra após a outra, não apresentando desta maneira nenhuma idéia de recursividade. A recursividade é caracterizada quando o processamento da ação de uma determinada regra ativa R, gera um evento que dispara a mesma regra R. Em outras palavras, este algoritmo, aparentemente muito simples, esconde em verdade uma grande complexidade [CW96].

Ações de regras podem disparar novas regras gerando assim um *loop*, muitas vezes, infinito, independente de se ter um algoritmo iterativo ou recursivo, sequencial ou concorrente. É necessário definir um **ponto de parada** para o algoritmo de processamento de regras apresentado acima. Se, pela sintaxe da linguagem de regras e pelas definições das regras existentes, existir a certeza de que nunca será gerado um caso de *loop* infinito dentro da execução de regras do BD, pode-se eliminar a preocupação com um ponto de parada. Porém, se esta certeza não existir, o ponto de parada do algoritmo deve ser definido, por exemplo, levando-se em consideração o número pré-definido de regras que possam ser disparadas após o início do processamento de regras. Neste caso, se um número maior de regras forem disparadas, o programa termina anormalmente.

A seguir são detalhados os diferentes aspectos da semântica de execução de regras ativas.

Na maioria dos casos em que os eventos são operações de manipulação do BD, a execução de uma regra ativa pode ser **orientada-a-instância** ou então **orientada-a-conjunto**. No primeiro caso, a um evento relacionado a uma operação sobre uma instância de um objeto do BD, por exemplo, um registro de uma tabela, pode ser disparada uma regra ativa associada a esse evento. Na execução orientada-a-conjunto, o evento está relacionado ao acontecimento de um conjunto de operações sobre o BD, como por exemplo, a inserção de várias linhas em uma tabela do BD. O elemento definido aqui é denominado de **granularidade**, dentro da semântica de execução de regras.

Um outro elemento da semântica de execução de regras é a **política de detecção e solução de conflitos**. Uma situação de conflito pode ocorrer se várias regras são disparadas pelo mesmo evento, e se as regras disparadas interferem umas com as outras. Quando ocorre uma situação de conflito, torna-se necessário definir uma política de solução de conflitos. São exemplos de políticas adotadas para a solução de conflitos no processamento de regras:

- escolha arbitrária de uma regra para ser processada;

- escolha de uma regra baseada na especificação de prioridades, definidas no momento de definição das regras;
- escolha de uma regra baseada em propriedades estáticas da regra, como tempo de criação da regra no BD;
- escolha de uma regra baseada em propriedades dinâmicas da regra, como quão recentemente essa regra foi disparada.

O relacionamento entre a ocorrência de um evento e o disparo da regra, ou entre a avaliação da condição de uma regra e o processamento de sua ação, são chamados, respectivamente, de **acoplamento evento-condição (E-C)** e **acoplamento condição-ação (C-A)** [ZSF+98]. Cada modo de acoplamento pode ser qualificado da seguinte maneira:

- Acoplamento Imediato (*Immediate*): para acoplamentos E-C, Imediato define que a condição da regra é avaliada imediatamente após o evento. Para acoplamentos C-A, Imediato define que a ação da regra deve ser processada imediatamente após a avaliação da condição da regra, se esta for satisfeita;
- Acoplamento Postergado (*Deferred*): para acoplamentos E-C, Postergado define que a condição da regra é avaliada em algum momento bem definido após o evento. Para acoplamentos C-A, Postergado define que a ação da regra deve ser processada em algum momento bem definido após a avaliação da condição da regra, se esta for satisfeita.

Pode-se citar como exemplo de um sistema em que existem os dois modos de acoplamento o DICE – *Distributed and Integrated Cooperative Environment* [SN97].

Também faz parte da semântica de execução de regras **o relacionamento entre as regras ativas e as transações do BD**. Em geral, esses relacionamentos se dão da seguinte maneira:

- Relacionamento Dependente: neste caso, a regra disparada por um evento é processada na mesma transação do evento (*triggering transaction*);
- Relacionamento Independente: neste caso, a regra dispararada por um evento

é processada em uma transação independente (*triggered transaction*) da transação do evento.

Pode-se concluir que são muitas as funcionalidades, sintáticas e semânticas, a serem oferecidas por um SGBDA. A presente proposta não ambiciona explorar todas essas funcionalidades mas selecionar algumas delas e implementá-las facilmente em SGBDs que não disponham delas. Para que a implementação seja fácil, é preciso examinar algumas arquiteturas de transformação de SDGBDs passivos em SGBDs ativos.

## 2.2 Arquiteturas de Transformação de SGBDs Passivos em Ativos

Pode-se melhorar as funcionalidades ativas de um SGBD de várias maneiras. Neste trabalho, entretanto, são analisadas apenas duas propostas de arquitetura mais usadas.

### 2.2.1 Arquitetura em Camadas

A primeira proposta é a construção de uma camada ativa que sirva de interface entre o SGBD passivo ou semi-ativo (que serão chamados simplesmente de SGBD Base) e as aplicações do usuário (Figura 2.1). A arquitetura em camadas permite que a camada ativa ofereça novas funcionalidades ativas, reutilizando o código do SGBD Base, evitando, dessa maneira, quaisquer alterações no seu código fonte[CW96].

O gerenciamento de eventos é feito através da interceptação, pela camada ativa, dos comandos submetidos ao SGBD e dos dados que são retornados para a aplicação ou usuário.

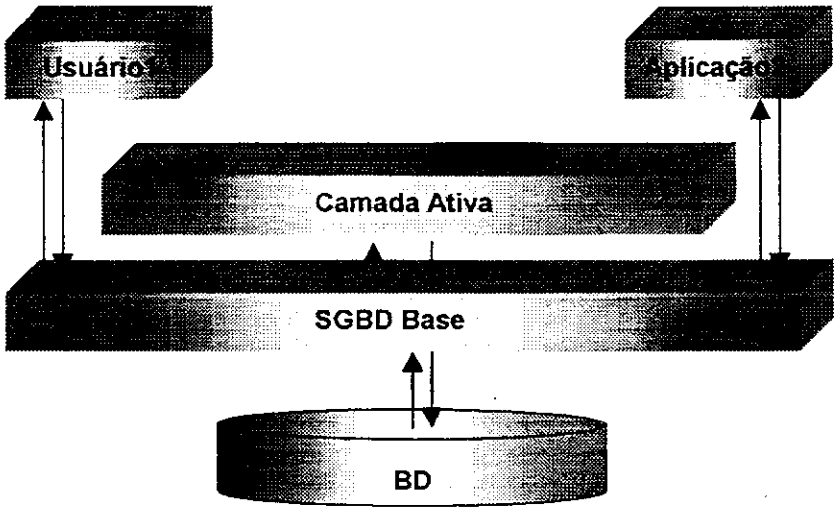



**Figura 2.1 – Arquitetura em Camadas**

As vantagens da arquitetura em camadas, além de ser transparente ao usuário comum (note que o usuário comum interage com o SGBD e não com a camada ativa), é que se consegue implementar um SGBD Ativo a partir de um SGBD Base sem, praticamente, nenhuma alteração no código do SGBD Base. De outra forma, ficaria muito mais difícil estender as funcionalidades ativas de um SGBD existente. Também, com uma camada ativa pode-se padronizar as funcionalidades ativas; mais precisamente, dados vários SGBDs Base, pode-se dotar todos eles das mesmas funcionalidades ativas, uma vez que a camada ativa é comum a todos eles [GKBF98].

Como desvantagens desse tipo de arquitetura, pode-se citar a baixa performance devido ao *overhead* criado pelo excesso de comunicação entre o módulo ativo e o SGBD Base. Outra desvantagem é que, como a camada ativa é uma aplicação normal do SGBD Base, ficaria muito difícil adicionar à mesma certas funcionalidades ativas. Por exemplo, se o SGBD Base tratar cada regra ativa como uma transação independente, será extremamente complicado implementar uma nova funcionalidade, como fazer uma regra ativa parte de uma outra transação.

### 2.2.2 Arquitetura *built-in*

A arquitetura *built-in* é outra forma de se conseguir um SGBD Ativo partindo de

um SGBD Base. Esta arquitetura consiste em modificar o código do SGBD Base, como vemos na Figura 2.2, para implementar as facilidades ativas [CW96].

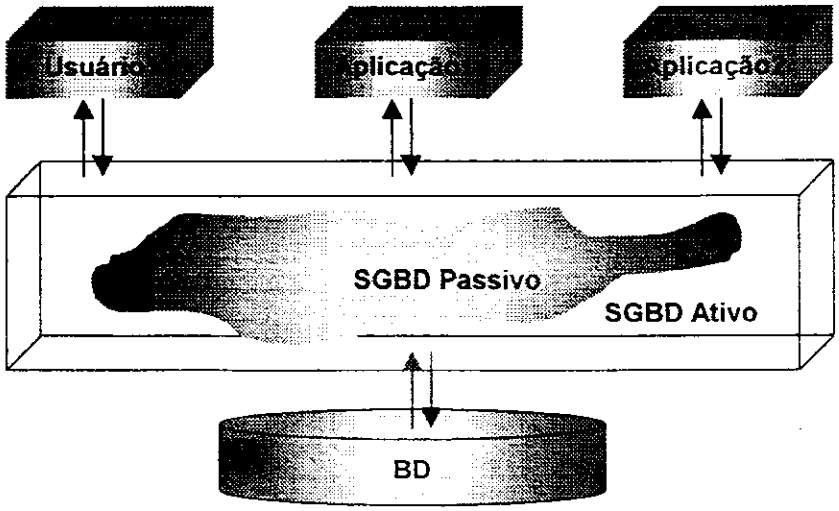



**Figura 2.2 – Arquitetura *built-in***

Nesta arquitetura, o gerenciamento e o processamento das regras são integradas ao sistema existente. Quando ocorre um evento, o próprio SGBD se encarrega de "avisar" o seu "módulo ativo" que gerencia internamente todo o processamento de regras, de acordo com a semântica especificada.

Pode-se observar que as vantagens e desvantagens da arquitetura *built-in* praticamente se invertem quando comparadas com a arquitetura em camadas. Como vantagem principal pode-se citar o fato de que a semântica de execução de regras pode se tornar muito mais sofisticada, pois o monitoramento de regras, a avaliação das condições e a execução das ações fazem parte da especificação do próprio SGBD.

As desvantagens se tornam evidentes quando se deseja alterar o código-fonte de um SGBD para estender suas funcionalidades ativas. Além de muito complicado, se isto fosse possível, cada SGBD estendido teria funcionalidades ativas bem específicas, em detrimento da padronização.

Como conclusão, apesar de algumas desvantagens da arquitetura em camadas, esta ainda se torna muito atraente em comparação com a arquitetura *built-in* pelos motivos explicados. Desta forma, a arquitetura em camadas foi a escolhida para servir

como base da presente proposta.

## 2.3 Implementação Bancos de Dados Ativos: Estado da Arte

Esta seção tem a função de apresentar as funcionalidades ativas dos padrões SQL92 e SQL3, da arquitetura *Alert*, e de alguns dos principais SGBD comerciais, terminando por uma síntese comparativa baseada na taxonomia proposta em [CW96].

### 2.3.1 Padrões e Arquiteturas

São sucintamente apresentadas, nesta sub-seção, as propostas de funcionalidades ativas dos padrões relacionais SQL92 e SQL3, e uma interessante arquitetura chamada *Alert*.

#### O Padrão SQL-92

Apesar de não apresentar funcionalidades ativas propriamente ditas, o padrão SQL-92 [MS93][CO92] implementa o conceito de restrições de integridade que serviu de base para o conceito de *triggers*, funcionalidade ativa do padrão SQL3, apresentado em seguida. O SQL-92 suporta uma rica coleção de restrições que podem ser classificadas em três categorias: restrições de domínio, restrições de integridade referencial e asserções.

Restrições de Domínio

São restrições usadas para garantir que dados inseridos em uma determinada coluna de uma determinada tabela pertençam a um determinado domínio[2], isto é, respeitem certas características especificadas no instante de criação da tabela[KS93]. Uma coluna de uma tabela definida com a restrição *NOT NULL*, por exemplo, não permite valores nulos naquela coluna, em nenhum registro da tabela. Seja `filmes` uma tabela de um BD, criada com o comando SQL `CREATE TABLE`, com as seguintes colunas: `nome_filme` e `nome_diretor`, sendo a coluna `nome_filme` definida com a restrição

[2] - Conjunto de valores válidos para uma determinada coluna.

*NOT NULL:*

```
CREATE TABLE filmes
(nome_filme CHARACTER (30) NOT NULL,
 nome_diretor CHARACTER (30) );
```

A restrição *NOT NULL* não permite que um registro do tipo (Null,'Francis Ford Copolla') seja inserido na tabela `filmes`. Note que a implementação desta restrição é uma regra ECA implícita onde, o evento é caracterizado pela inserção de um registro na tabela `filmes`, a condição é que a coluna `nome_filme` seja preenchida com um valor não nulo e a ação é não permitir que o registro seja inserido, caracterizando assim uma situação de erro.

Restrições de Integridade Referencial

Restrições de Integridade Referencial envolvem duas tabelas, denominadas de tabela que faz referência e tabela referenciada. A tabela que faz referência possui uma ou mais colunas com o mesmo domínio da(s) coluna(s) chave(s) primária(s)[3] da tabela referenciada. Por exemplo, dadas as tabelas `empregado` e `departamento` e a condição de que todo empregado deve estar lotado em um departamento, a tabela `departamento` deve ter colunas como `cod_depto` (chave primária) e `nome_depto`. Por sua vez, a tabela `empregado` deve ter uma coluna como `cod_depto` (chave estrangeira[4] ou *foreign key*) que permite associar cada empregado a seu respectivo departamento. Neste exemplo, a tabela que faz referência é a tabela `empregado` enquanto que `departamento` é a tabela referenciada.

Uma restrição de integridade referencial é implementada por uma regra ECA implícita, do tipo:

```
<foreing key action> ::= <event><action>
<event> ::= ON UPDATE | ON DELETE
<action> ::= CASCADE | SET DEFAULT | SET NULL |
             NO ACTION;
```

[3] - Coluna(s) que identifica(m) de forma única um registro de uma tabela.

[4] - Coluna que referencia outra tabela.

Vamos dar apenas o exemplo de um departamento que é excluído da tabela `departamento` e a ação escolhida é `cascade`. Neste caso, todos os registros de `empregado` em que os valores da coluna `cod_depto` seja o departamento excluído serão também excluídos, automaticamente. Mais informações sobre integridade referencial podem ser obtidas em [D81].

Asserções

Asserções são restrições gerais que podem envolver múltiplas tabelas. A sintaxe de asserções é:

```
<SQL92_assertion> ::= CREATE ASSERTION <constraint_name>
                      CHECK (<condition>)
                      [<constraint_evaluation>]
```

Uma asserção é violada se a condição da cláusula `CHECK` for satisfeita. O leitor encontrará uma discussão detalhada de asserções em [MS93].

**O Padrão SQL3**

O conceito fundamental do padrão SQL3 é o de *trigger*. *Triggers* (ou gatilhos) se adequam ao conceito de regras ECA. O evento de um *trigger* é uma operação de modificação executada sobre uma tabela, a condição é um predicado SQL definido na cláusula `WHEN`, e a ação é um conjunto de comandos SQL. Cada *trigger* reage a uma operação de modificação de dados sobre uma tabela, a ação acontecendo antes, depois ou no lugar da operação que caracterizou o evento(cláusulas `BEFORE`, *`AFTER`* ou `INSTEAD OF` respectivamente). A sintaxe completa de um *trigger* é a seguinte:

```
<SQL3_trigger> ::= CREATE TRIGGER <trigger_name>
      {BEFORE | AFTER | INSTEAD OF} <trigger_event>
      ON <table_name>
      [ORDER <order_value>]
      [REFERENCING <reference>]
      WHEN (<condition>)
      <SQL_procedure_statements>
```

```
        <FOR EACH {ROW | STATEMENT}]

<trigger_event> ::= INSERT | DELETE | UPDATE
        [OF <columns_names>]

<reference> ::= OLD AS <old_value_tuple_name> |
        NEW AS <new_value_tuple_name> |
        OLD_TABLE AS <old_value_table_name> |
        NEW_TABLE AS <new_value_table_name>
```

Segue-se uma breve explanação da semântica de execução de um *trigger* no padrão SQL3.

Se um *trigger* é orientado-a-instância então utiliza-se a cláusula `FOR EACH ROW`. Se, por outro lado, um *trigger* é orientado-a-conjunto então utiliza-se a cláusula `FOR EACH STATEMENT`.

O SQL3 permite definir que a ação do *trigger* substitua a operação do evento além da definição de antes e depois. A cláusula `INSTEAD OF` é a responsável por essa facilidade.

Para a solução de conflitos, utiliza-se a cláusula `ORDER`.

Outra funcionalidade ativa importante e presente no padrão SQL3 é a possibilidade de manipulação de valores de transição usando a cláusula `REFERENCING`. Os valores de transição antigo e novo são referenciados pelas cláusulas `OLD` e `NEW` em caso de `FOR EACH ROW` e pelas cláusulas `OLD_TABLE` e `NEW_TABLE` em caso de `FOR EACH STATEMENT`.

**A Arquitetura *Alert***

*Alert* [SPAM91] é uma camada ativa sobre o SGBD *Starburst* da IBM [HCL+90], que suporta a implementação de múltiplas linguagens de regras com sintaxe e semântica diferentes, visando estender as funcionalidades ativas do SGBD Base citado (Figura 2.3).

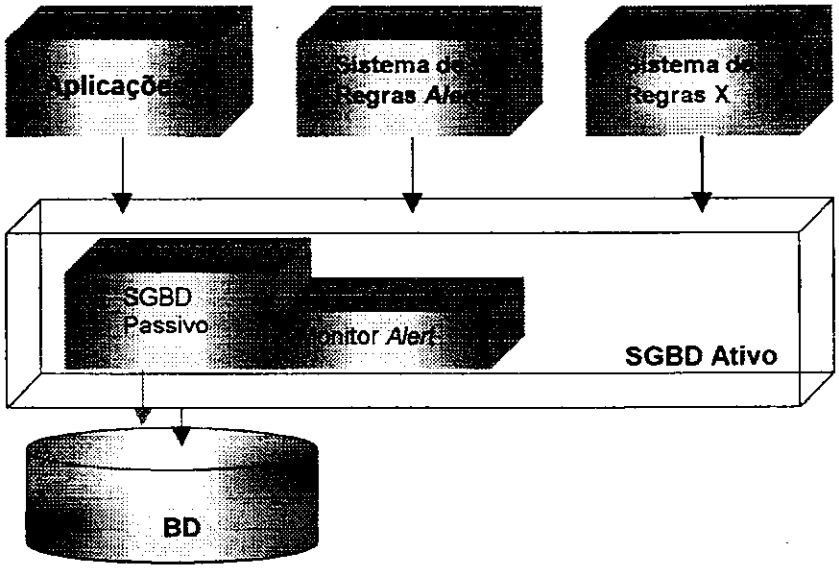



**Figura 2.3 – Arquitetura *Alert***

A arquitetura é composta basicamente de duas camadas, uma camada baixa, chamada de Monitor, responsável pelo gerenciamento das funcionalidades ativas e que "conversa" diretamente com o SGBD Base; e uma camada alta, formada por várias linguagens de regras, que usam os serviços do Monitor, e por aplicações que acessam a camada ativa de maneira transparente.

Uma das linguagens de regras implementadas na *Alert* é *SQL-like*, oferecendo a vantagem de manipular regras ativas com um mínimo de alterações na linguagem SQL básica [D89].

A arquitetura *Alert* é baseada nos conceitos de Tabelas Ativas e Consultas Ativas para implementar o mecanismo de detecção de eventos.

O conceito de consultas ativas é semelhante ao de consultas passivas, a diferença estando no comportamento do cursor-SQL [MS93]. Em uma consulta passiva, quando um cursor é aberto, operações `fetch` sucessivas buscam registros que satisfaçam a consulta até encontrar um EOF (*end-of-file*). Uma nova consulta repete todo o processo, para os novos registros selecionados. Já as consultas ativas são definidas em cima de tabelas ativas, que são tabelas *append-only*, isto é, tabelas onde novos registros

sempre são adicionados no final. Assim, o cursor para uma tabela ativa é aberto uma única vez. Para isto, o sistema *Alert* introduziu uma nova primitiva `fetch-wait`, que permite que o cursor fique em estado de espera por novos registros. Quando uma tabela ativa é criada, nela é documentada, automaticamente, toda operação definida como um evento.

Seja um sistema de controle de caldeiras. A tabela ativa `caldeira`, criada com o comando `create active table caldeira(id_caldeira integer, temp_lida real, hora time)`[5], serve de cadastro de leitura da temperatura das caldeiras existentes. Esta leitura é feita a cada 60 segundos por um sensor e este sensor ativa um procedimento que gera os registros da tabela `caldeira`. Suponhamos que este procedimento seja denominado de `leitura_caldeira`. Define-se então um cursor `C` sobre a tabela ativa `caldeira`:

**Exemplo 2.1:**

```
declare C cursor for
      SELECT id FROM caldeira
      WHERE temp_lida > 100
open C;
      while (TRUE)
      { fetch-wait  C into :id;
      ...
      }
```

A consulta acima recupera todos os registros já incluídos na tabela `caldeira` satisfazendo a condição dada e entra em estado de espera até que novos registros sejam incluídos por `leitura_caldeira` na tabela ativa `caldeira`.

As regras *Alert* são definidas especificando as condições da regra nas cláusulas `FROM` e `WHERE` (SQL) e as ações na cláusula `SELECT` (*SQL-like*, portanto). O evento, como visto anteriormente, é implícito (uma inclusão de um registro na tabela ativa).

[5] - Note que este comando é uma extensão do SQL básico.

Por exemplo, seja `RegraDesligamentoCaldeira` uma regra definida para a tabela ativa `caldeira`. A regra `RegraDesligamentoCaldeira` ativa um procedimento de desligamento do forno da caldeira quando temperaturas acima do limite máximo suportado são atingidas, evitando assim a explosão da caldeira. Este procedimento poderá ser denominado `desliga`. A definição da regra `RegraDesligamentoCaldeira` fica:

**Exemplo 2.2:**

```
Create rule RegraDesligamentoCaldeira as
      SELECT desliga( id_caldeira )
      FROM caldeira
      WHERE temp_lida > 1000
```

onde: `id_caldeira, temp_lida` - são campos da tabela ativa `caldeira`.

Como as regras da *Alert* são *SQL-Like*, elas se beneficiam de todo o poder de expressão dessa linguagem, suportando uma grande variedade de construções. Porém, elas têm muitas limitações, entre outras:

- não permitem a definição de eventos sobre mais de uma tabela;
- não permitem a definição de eventos compostos;
- não permitem definir se o processamento da regra será feito antes ou depois da ocorrência do evento.

Em relação à semântica de execução de regras-*Alert*, dois modos de acoplamento podem ser especificados *Immediate* e *Deferred*; dois tipos de relacionamentos com transações também podem ser especificados *Same* e *Separate*, tirando proveito de características da implementação do Starburst[SPAM91].

### 2.3.2 SGBDs Comerciais

A maioria dos SGBDs comerciais atualmente implementa alguma variação do conceito de *trigger-SQL3*. A seguir, são detalhados os *triggers* dos principais SGBDs comerciais.

**Oracle 7.3**

No que diz respeito às funcionalidades ativas do Oracle7, ele implementa um conceito de *trigger* (*trigger-Oracle*) como procedimentos armazenados no BD e relacionados a uma única tabela-Oracle, sendo implícita e automaticamente disparados pelo Oracle quando as tabelas associadas respectivas são modificadas.

Um *trigger-Oracle* é composto de três partes: o evento, a condição e a ação. Os comandos INSERT, DELETE e UPDATE da linguagem SQL formam o conjunto de eventos suportados pelo Oracle. No caso da operação UPDATE, o Oracle permite especificar as colunas da tabela que, se forem alteradas, caracterizarão um evento. O Oracle permite ainda uma forma limitada de definição de múltiplas operações na especificação do evento de um *trigger*. A sintaxe de definição de *triggers* no Oracle 7.3 é a seguinte:

```
<trigger_oracle> ::= {CREATE | REPLACE} TRIGGER <nome_trigger >
                     {BEFORE | AFTER} <evento_trigger>
                     {ON <nome_tabela> [[REFERENCING <referencia>]
                     FOR EACH ROW
                     [WHEN <condicao>]]
                     <PL/SQL_block>

<evento_trigger> ::= <comando> [OR <comando>]

<comando> ::= INSERT | DELETE | UPDATE [OF <column_name>]

<reference> ::= OLD AS <old_values_tuple_name> |
                NEW AS <new_value_tuple_name>
```

Note que o Oracle implementa uma forma primitiva de evento composto envolvendo apenas o operador *OR*, ou seja, na prática eventos no Oracle7 são simples.

A restrição (cláusula WHEN) é definida como um predicado simples sobre o

registro modificado e só é possível ser definida em conjunção com a opção FOR EACH ROW.

A ação de um *trigger* é um procedimento que contém instruções SQL e/ou PL/SQL, podendo definir inclusive construções como variáveis, constantes, cursores, da linguagem PL/SQL e suportando a chamada a *stored procedures*. A ação do *trigger* também pode referenciar valores de transição de registros.

Em relação à semântica de execução de regras, no Oracle7 o processamento dos *triggers* é feito em uma transação diferente da que gerou o evento.

**Informix 6.0**

No Informix 6.0 [Inf94] pode existir mais de um par (Condição, Ação) para cada evento definido em um único *trigger*, ou melhor, pode ser definido um par (Condição, Ação) para ser disparado antes do evento (cláusula BEFORE), um par para ser disparado a cada linha afetada pelo comando gerador do evento (cláusula FOR EACH ROW) e um par para ser disparado depois do processamento do evento (cláusula AFTER). Essa característica difere do Oracle7 que só permite definir um par (Condição, Ação) por *trigger* definido.

A sintaxe de definição de *triggers* no Informix 6.0 é a seguinte:

```
<Informix_trigger> ::= CREATE TRIGGER <trigger_name>
            <trigger_event> ON <table_name>
            [BEFORE [<condition>] (<actions>)]
            [[REFERENCING <references>]
            FOR EACH ROW [<condition>] (<actions>)]
            [AFTER [<condition>] (<actions>)]

<trigger_event> ::= INSERT | DELETE | UPDATE
            [OF <column names>]

<reference> ::= OLD AS <old_value_tuple_name> |
            NEW AS <new_value_tuple_name>
```

```
<condition> ::= WHEN (<predicate>)

<action> ::= <insert_statement> | <delete_statement>
             <update_statement> | <procedure call>
```

A condição da cláusula WHEN pode ser um predicado arbitrário ou uma chamada a uma *stored procedure*, se for omitida, implicitamente retorna *TRUE*. As ações podem ser uma sequência arbitrária de comandos INSERT, DELETE e/ou UPDATE, bem como chamadas a *stored procedures* (com a condição de que estas *stored procedures* não envolvam comandos de controle de transações).

Quanto a semântica de execução de regras, o Informix 6.0 permite a manipulação de valores de transição através das palavras reservadas OLD AS e NEW AS que referenciam os valores antigos e novos, respectivamente.

**Ingres 6.4**

O SGBD Ingres 6.4 suporta *triggers* denominados de regras, como parte do *Knowledge Management Extension* [CW96]. Os comandos INSERT, DELETE ou UPDATE funcionam como evento disparador e a ação da regra é representada por uma chamada a um procedimento. Não permitem nenhuma forma de composição de eventos. A sintaxe da definição de regras no Ingres 6.4 é a seguinte [ASK92]:

```
<Ingres_rule> ::= CREATE RULE <rule_name>
                  AFTER <rule events> ON <table_name>
                  [REFERENCING <references>]
                  WHERE <predicate>
                  EXECUTE PROCEDURE <procedure_call>

<rule_event> ::= INSERT | DELETE | UPDATE [(<column_name>)]

<reference> ::= OLD AS <old_value_tuple_name> |
                NEW AS <new_value_tuple_name>
```

Em relação a semântica de execução de regras o Ingres 6.4 permite apenas que as regras sejam executadas após o processamento do evento disparador da regra e que

sejam orintadas-a-instância. Permite referenciar valores de transição, usando as cláusulas OLD e NEW, do registro que está sendo manipulado além de constantes.

**Sybase Server Release 10.0**

Como no Ingres 6.4, os *triggers* do Sybase 10.0 também só podem ser executados após a execução do evento disparador [Syb]. Podem ser definidos até três *triggers* para cada tabela, um para cada operação: INSERT, DELETE ou UPDATE [CW96]. Duas variantes são permitidas na sintaxe de definição de *triggers* no Sybase:

```
<Sybase_trigger1> ::= CREATE TRIGGER <trigger_name>
                      ON <table_name>
                      FOR <trigger_events>
                      AS <SQL_statements>

<trigger_event> ::= INSERT | DELETE | UPDATE

<Sybase_trigger2> ::= CREATE TRIGGER <trigger_name>
                      ON <table_name>
                      FOR <trigger_restricted_events>
                      AS [IF <trig_upd_exp>] <SQL_statements>

<trigger_restricted_events> ::= INSERT | UPDATE

<trig_upd_exp> ::= UPDATE (<column_name>) |
                      <trig_upd_exp> AND <trig_upd_exp> |
                      <trig_upd_exp> OR <trig_upd_exp>
```

As condições não são definidas explicitamente, mas se necessário, elas podem ser definidas no corpo da ação relacionada. Outra limitação é a impossibilidade de se usar alguns tipos de comandos no corpo da ação como por exemplo, comandos de definição de dados e de consulta (SELECT). A vantagem em relação ao Ingres 6.4 é que comandos como ROLLBACK TRANSACTION e ROLLBACK TRIGGER são permitidos.

A semântica de execução de regras adotada permite que o tratamento de valores de transição seja feito através de duas tabelas temporárias definidas pelo sistema, chamadas INSERTED e DELETED. Os valores relacionados a UPDATE são tratados como

um `DELETE` seguido por um `INSERT`.

### O Projeto HiPAC

Apesar de não ter se tornado um produto comercial, o projeto HiPAC [CBB+89] foi um dos primeiros projetos na área de Bancos de Dados Ativos Orientado a Objetos. As características ativas são baseadas no conceito das regras ECA, considerando eventos temporais e incluindo facilidades para a especificação de restrições de tempo como parte da linguagem de regras. A linguagem de regras e o modelo de execução do HiPAC, refletem uma clara separação entre eventos, condições e ações [CW96].

Regras ativas no HiPAC são tratadas como objetos de primeira classe, podendo ser criadas, alteradas ou destruídas como qualquer outro objeto.

Os eventos também são tratados como entidades de primeira classe e podem ser simples ou compostos. Os eventos simples podem ser eventos de manipulação de dados, temporais (absoluto, relativo ou periódico) e/ou de notificação externa (sinalizado explicitamente pelo usuário ou por um programa de aplicação). Os eventos compostos podem ser de três tipos: disjunção (sejam E1 e E2, eventos, a regra associada é disparada se E1 ocorrer ou E2 ocorrer), sequência (a regra associada só é disparada se E2 ocorrer depois de E1) ou fechamento (a regra associada só é disparada se E1 ocorrer mais de uma vez dentro de um determinado período de tempo).

A semântica de execução de regras adotada pelo HiPAC possibilita definir um modo de acoplamento para o par (Evento, Condição) e outro para o par (Condição, Ação). São suportados três tipos de modos de acoplamento:

- imediato: para o par (Evento, Condição), faz a condição ser avaliada imediatamente após o evento ser detectado; para o par (Condição, Ação), faz a ação ser executada logo após a condição ter sido avaliada;
- retardado: para o par (Evento, Condição), faz a condição ser avaliada após a última operação da transação que gerou o evento, ter sido concluída; para o

par (Condição, Ação), faz a ação ser executada após a última operação da transação que avaliou a condição, ter sido concluída;

- desacoplado: para o par (Evento, Condição), faz a condição ser avaliada em uma transação diferente da que gerou o evento; para o par (Condição, Ação), faz a ação ser executada em uma transação diferente da que avaliou a condição.

Um evento pode disparar mais de uma regra no projeto HiPAC. A execução das regras disparadas é serializável e pode usar três mecanismos de resolução de conflitos:

- níveis de prioridade: associados às regras;
- mecanismo de ciclo: que ordena a execução de regras retardadas;
- mecanismo de *pipelining*: usado para o modo de acoplamento desacoplado, permite que a ordem de execução das transações disparadas reflitam a ordem das transações que as dispararam.

Apesar de se apresentar como um verdadeiro SGBDA, o HiPAC, como foi dito no início da seção, não é um produto comercial, não estando disponível assim para comercialização.

Na seção seguinte é apresentada uma tabela comparativa entre as características ativas apresentadas pelos padrões SQL92 e SQL3, pela arquitetura *Alert* e pelos SGBD vistos nesta seção.

## 2.4 Análise Comparativa das Características Ativas Apresentadas pelos SGBDs Estudados

Existem uma série de funcionalidades ativas que devem estar presentes em um SGBD para este ser considerado um verdadeiro SGBDA [CW96]. Tomando como base a presença ou não destas funcionalidades, os principais padrões, arquiteturas e SGBDs comerciais foram estudados e classificados. A seguir é apresentada uma tabela comparativa das funcionalidades apresentadas por cada padrão, arquitetura e SGBDs estudados.

| | PADRÕES | | ARQ | SGBDRs | | | | SGBDOO |
|---|---|---|---|---|---|---|---|---|
| **EVENTOS** | **SQL-92** | **SQL3** | ***Alert*** | **Oracle** | **Informix** | **Ingres** | Sybase | HiPAC |
| Modificação de Dados | X | X | X | X | X | X | X | X |
| Recuperação de Dados | - | - | - | - | - | - | - | X |
| Temporais | - | - | - | - | - | - | - | X |
| Definidos pela Aplicação | - | - | X | - | - | - | - | X |
| Disparos | | | | | | | | |
| Antes | - | X | - | X | X | - | - | X |
| Depois | - | X | - | X | X | X | X | X |
| Compostos | | | | | | | | |
| Operadores Lógicos | - | - | - | X | - | - | - | X |
| Sequência | - | - | - | - | - | - | - | X |
| Composição Temporal | - | - | - | - | - | - | - | - |
| Parametrizados | - | - | - | - | - | - | - | X |
| **CONDIÇÕES** | | | | | | | | |
| Predicados do BD | X | X | X | - | X | X | - | X |
| Predicados Restritos | - | - | - | - | - | - | - | - |
| Consultas ao BD | - | - | X | X | - | - | - | - |
| Procedimentos da aplicação | - | - | X | - | X | - | - | X |
| **AÇÕES** | | | | | | | | |
| Modificação de Dados | X | X | X | X | X | - | X | X |
| Recuperação de Dados | - | X | X | X | - | - | - | X |
| Outros Comandos do BD | - | - | - | X | - | - | X | - |
| Procedimentos da Aplicação | - | - | X | X | X | X | X | X |
| **OUTRAS CARACTERÍSTICAS** | | | | | | | | |
| Valores de Transição | X | X | - | X | X | X | X | X |
| Eventos Lógicos | - | - | - | X | - | - | - | - |
| Comandos para Regras | - | X | X | X | X | X | X | X |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Prioridades de Regras | - | X | - | - | - | - | - | X |
| Estruturação de Regras | - | - | X | - | X | X | X | - |
| Regras Entidades 1a classe | - | X | X | X | - | - | - | X |
| Granularidade | | | | | | | | |
| Orientada a Instância | - | X | - | X | X | X | - | X |
| Orientada a Conjunto | X | X | - | X | - | - | X | X |

**Tabela 2.1 - Análise Comparativa das Características Ativas Apresentadas pelos SGBDs Estudados**

No âmbito da definição de eventos, nota-se que existe uma precariedade de funcionalidades ativas oferecidas. Com exceção do HiPAC, que não é um SGBD comercial, nenhum dos outros SGBDs analisados suportam a definição de eventos do tipo recuperação de dados, temporais ou parametrizados. Com exceção também do HiPAC, nenhum oferece a capacidade de definir verdadeiramente eventos compostos. O Oracle 7.3 apresenta uma forma muito primitiva de composição de eventos utilizando apenas o operador lógico *OR*. Com exceção da arquitetura *Alert*, que é especifica para o protótipo Starburst da IBM, e do HiPAC, nenhum implementa eventos definidos pela aplicação. Os padrões SQL92 e SQL3 apresentam funcionalidades ativas muito primitivas.

Para suprir boa parte dessas deficiências, surge a proposta do Engenho Ativo, especificado no próximo capítulo.

# 3

# Objetivos e Especificação do Engenho Ativo

Neste capítulo, é apresentado o Engenho Ativo, que é uma camada ativa implementada no topo de um SGBD Relacional e que visa dotá-lo de algumas funcionalidades ativas importantes não contempladas por ele.

## 3.1 Objetivos do Engenho Ativo

Como foi discutido no capítulo anterior, a maioria dos SGBDs comerciais oferece funcionalidades ativas, porém com muitas limitações. O objetivo do Engenho Ativo é, dado um SGBD comercial ou SGBD Base, estender suas funcionalidades ativas, sem depender para isto do código-fonte do SGBD Base. O Engenho Ativo oferece ao

usuário, ou mais precisamente, ao Administrador de Banco de Dados - ABD, uma interface simples e poderosa para manipular regras ativas.

Como pontos fortes do Engenho Ativo pode-se citar:

a) a capacidade de definir procedimentos do usuário como geradores de eventos;
b) a possibilidade de definir eventos compostos;
c) a capacidade de definir eventos envolvendo várias tabelas de um BD Relacional;
d) a possibilidade de fazer chamadas a procedimentos de usuário como parte da condição da regra;
e) a possibilidade de definir ações como chamada a qualquer rotina executável, independentemente do seu código-fonte.

É importante notar que as funcionalidades a) e c) não são oferecidas pelos SGBDs comerciais e que as funcionalidades b) e e) o são de forma limitada. O Oracle7 por exemplo, implementa uma forma limitada de definição de eventos compostos, a composição sendo feita via operador *OR*; assim, para uma expressão de eventos ser satisfeita basta um dos operandos assumir o valor VERDADE, o que não é suficiente para caracterizar uma verdadeira composição de eventos.

A capacidade de fazer chamadas a procedimentos do usuário como parte da condição de uma regra (funcionalidade d)) é permitida, também de forma limitada, pelos SGBDs Oracle7 e Informix 6.0, por exemplo. A limitação se dá por conta do tipo de procedimento aceito como parte da condição. Só são permitidos procedimentos do tipo *stored procedures*[6] proprietárias, enquanto que o Engenho Ativo foi projetado para executar funções escritas em qualquer linguagem.

[6] - Uma *stored procedure* é um procedimento escrito em uma linguagem hospedeira de SQL de um SGBD, armazenado dentro do BD.

No final do capítulo, é apresentado um quadro comparativo das funcionalidades do Engenho Ativo com aquelas dos principais SGBDs comerciais.

## 3.2 Especificação Formal do Engenho Ativo

Nesta seção, é apresentado o conceito de regra ativa do Engenho Ativo, necessário para o bom entendimento do funcionamento do mesmo. São apresentadas também a linguagem de regras e a semântica de execução de regras do Engenho Ativo.

### 3.2.1 Regras Ativas

Regras Ativas no Engenho Ativo são, no geral, regras ECA (Evento, Condição, Ação). Mais detalhadamente, o Engenho Ativo trata como uma regra ativa o par (Condição, Ação), enquanto que Evento é a ocorrência disparadora (*"trigger"*) da regra ativa. Fica claro que, na definição de uma nova regra ativa, é necessário definir uma expressão de eventos para o seu disparo; nada impede, porém, de se criar mais de uma regra com a mesma expressão de eventos.

A seguir, detalharemos os conceitos de Evento, Condição e Ação do Engenho Ativo.

#### Evento

No Capítulo 2, foi definido como evento qualquer ocorrência que pode disparar uma regra ativa. É preciso desde já salientar que evento é *instantâneo*; assim, deve-se referir a um evento como (imediatamente) antes da ocorrência ou (imediatamente) depois da ocorrência.

O Engenho Ativo oferece a capacidade de definir como eventos, além de comandos SQL de manipulação de um BD (denominados de eventos-SQL), procedimentos definidos pelo usuário (eventos-procedimento-usuário) e expressões lógicas de eventos (eventos compostos).

Mais precisamente, eventos-SQL são eventos caracterizados pela ocorrência de comandos SQL INSERT, DELETE ou UPDATE, por exemplo, sobre uma tabela de um BD Relacional - BDR; eventos-procedimento-usuário são caracterizados por chamadas a procedimentos, em qualquer linguagem, escritos por usuários. Tanto eventos-SQL como eventos-procedimento-usuário são eventos *simples*. Eventos *compostos* são formados pela conjunção / disjunção / negação (operadores lógicos *AND*, *OR* e *NOT*, respectivamente) de eventos simples ou compostos.

Apesar de o Engenho Ativo não suportar o conceito de evento temporal, é notável o aumento do escopo das ocorrências que podem ser consideradas como evento a partir da possibilidade de se definir um procedimento de usuário como gerador de evento, além de eventos compostos. A título de ilustração, para definir uma consulta a uma determinada tabela de um BDR (SELECT em SQL) como geradora de um evento[7], basta criar um procedimento e no código do procedimento incluir esse comando SELECT.

A execução de uma regra ativa, na verdade a consideração da sua condição e a possível execução da sua ação, é realizada antes ou depois da ocorrência do evento disparador.

A definição de um evento disparador obedece à seguinte sintaxe:

```
<evento> ::= [ANTES | DEPOIS] (<expressao_evento>)

<expressao_evento> ::= <termo> OR (<expressao_evento>) | <termo>

<termo> ::= <fator> AND <termo> | <fator>

<fator> ::= <operando> | (<expressao_evento>) | NOT <fator>

<operando> ::= <procedimento_usuário>
```

Note que a especificação ANTES ou DEPOIS é opcional, significando que o valor *default* é DEPOIS.

[7] - Isto não é permitido nos SGBDs comerciais.

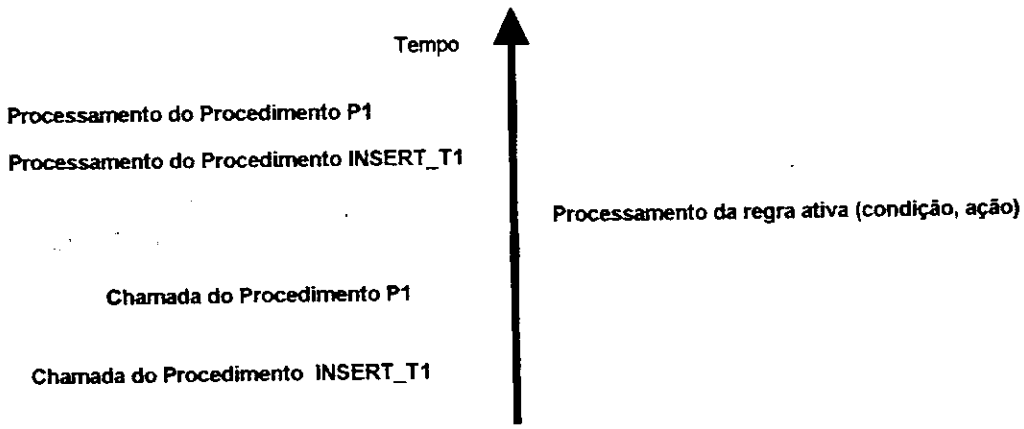
Supondo-se duas tabelas `T1` e `T2` de um BD, e um procedimento `P1`, é possível definir um evento da seguinte maneira:

**Exemplo 3.1:** `ANTES ((INSERT_T1 OR DELETE_T2) AND P1)`

onde a(s) regra(s) ativa(s) relacionada(s) ao evento deve(m) ser disparada(s) antes de acontecer a inserção de um registro em `T1` (chamada ao procedimento `INSERT_T1`) ou a remoção de um registro em `T2` (chamada ao procedimento `DELETE_T2`) e antes da chamada ao procedimento `P1`.

A Figura 3.1, Cenário 1, ilustra a semântica da cláusula `ANTES` na definição de um evento.

*Cenário 1* – `ANTES` *(<expressão_evento>)*



*Cenário 2* – DEPOIS *(<expressao_evento>)*

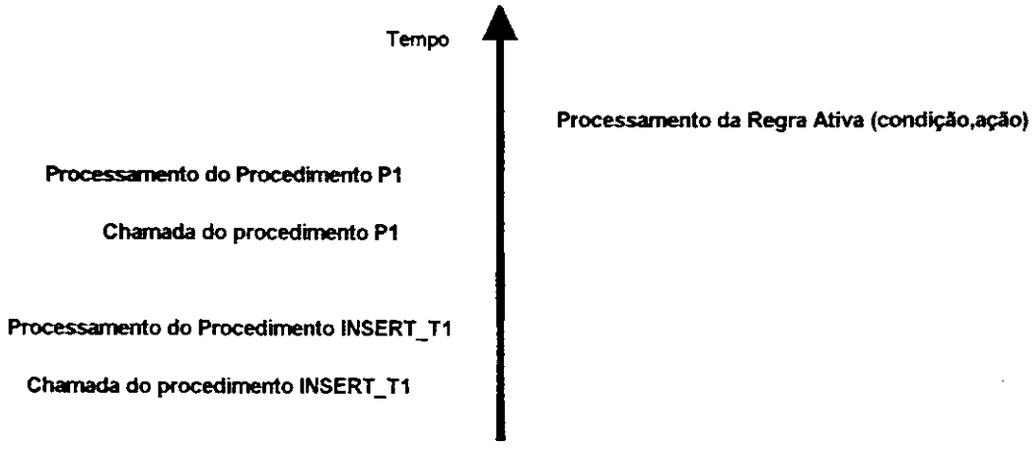



**Figura 3.1 – Semânticas de** ANTES **e** DEPOIS

A Figura 3.1, Cenário 2, ilustra a semântica de DEPOIS na definição de um evento.

### Condição

Uma condição funciona como um filtro para a execução da ação da regra ativa associada. Mais precisamente, quando ocorre um evento, a condição da regra ativa pertinente é considerada e, se satisfeita, a ação respectiva é executada. Uma condição pode ser definida de acordo com a seguinte sintaxe:

```
<expressao_condicao> → <termo> OR <expresso_condicao> |
                       <termo>
<termo> → <fator> AND <termo> | <fator>
<fator> → (<operando>) | (<expressao_condicao>) |
          (NOT <fator>)
<operando> →     <comando_manipulacao_SQL> |
                 CALL(<procedimento_usuário>) |
                 <consulta_ao_BD>
```

onde:

<comando_manipulacao_SQL>: comandos SQL INSERT, DELETE ou UPDATE que implicitamente retornam o valor VERDADE se for inserida, ou excluída,

ou alterada alguma tupla em uma determinada tabela, respectivamente; e um valor FALSO se nenhuma tupla for inserida ou excluída ou alterada;

`<procedimento_usuário>`: um procedimento do usuário neste contexto é uma função booleana (*stored procedure* ou procedimento externo ao BD);

`<consulta_ao_BD>`: comando SQL `SELECT` que implicitamente retorna o valor VERDADE se o conjunto-resposta da consulta não for vazio e o valor FALSO caso contrário.

A seguinte construção é aceita como uma expressão de condição válida na criação de uma regra:

**Exemplo 3.2:**
```
(SELECT campo1, campo2 FROM tabela1 WHERE
campo1 > 1000) AND (CALL (c:\usr\tricia\condicao.exe))
```

De acordo com o exemplo acima, a ação da(s) regra(s) ativa(s) associada(s) será(ão) executada(s) se a consulta `(SELECT campo1, campo2 FROM tabela1 WHERE campo1 > 1000)` selecionar alguma linha da tabela `tabela1` e se a rotina executável `condicao.exe`, presente no diretório `c:\usr\tricia`, retornar o valor booleano VERDADE. Note que, quando a condição é definida como um procedimento externo ao BD, como é o caso, deve ser inserido o *path* para que o Engenho Ativo consiga encontrá-lo.

**Ação**

A ação de uma regra ativa representa na realidade um conjunto de ações que são executadas como resposta a um determinado evento ocorrido, se a condição associada à regra ativa for satisfeita. Uma ação pode ser um conjunto de comandos de manipulação do BD e/ou chamadas a procedimentos do usuário.

A sintaxe para a ação de uma regra ativa é como se segue:

```
<acao> ::= <comando>;<comando>;...<comando>
<comando> ::=     <comando_SQL> |
                  CALL (<procedimento_usuario>)
```

```
<procedimento_usuario> ::= <stored_procedure> |
                           <arquivo>
```

onde:

`<comando_SQL>`: regido pela sintaxe e semântica da linguagem SQL;

`<arquivo>`: nome do arquivo incluindo o *path* e a extensão. No caso de extensões que não representem arquivos executáveis o arquivo é aberto no programa que o criou;

`<stored_procedure>` : procedimento armazenado no BD.

Seja uma regra ativa `R`, a ação de `R` sendo a seguinte:

**Exemplo 3.3:**
```
INSERT INTO tabela1 (campo1, campo2) VALUES (1,2);
CALL (c:\usr\tricia\acao.doc)
```

Mais claramente, se `R` for disparada e a condição de `R` for satisfeita, uma inserção de dados na tabela `tabela1` é feita, seguindo-se da apresentação do arquivo `acao.doc` pelo Word (extensão *.doc*).

### 3.2.2 Linguagem de Definição de Regras Ativas

O Engenho Ativo oferece ao usuário administrador de banco de dados uma interface de manipulação de regras ativas, compreendendo comandos para criar, excluir, ativar e desativar regras ativas. A seguir, são apresentados esses comandos, ilustrados com exemplos.

**Comando *Criar_Regra***

**Objetivo**: Criar uma regra ativa.

**Retorno**: Valor numérico. Assume os valores:

0 – Regra ativa criada com sucesso

1 – Erro, já existe uma regra ativa criada com o mesmo nome

2 – Erro de sintaxe

**Sintaxe**:

```
Criar_Regra ( <nome_regra>,
              <evento>,
              [<condicao>],
              <acao>,
              [<prioridade>]
            )
```

onde:

<nome_regra>: cadeia de caracteres com um tamanho máximo de 255 caracteres. Nomeia de forma única uma regra ativa;

<prioridade>: valor *default 0*;

**Exemplo 3.4**: A regra ativa R, constituída de suas partes:

Evento: `DEPOIS ((INSERT_T1 OR DELETE_T2) AND P1)`

Condição:
```
(SELECT campo1, campo2 FROM tabela1 WHERE campo1 > 1000)
    AND (CALL (c:\usr\tricia\condicao.exe))
```

Ação:
```
INSERT INTO tabela1 (campo1, campo2) VALUES (1,2);
    CALL (c:\usr\tricia\acao.doc)
```

e com a prioridade 1 é criada da seguinte maneira:

```
Criar_Regra( 'R',
'DEPOIS ((INSERT_T1 OR DELETE_T2) AND P1)',
'(SELECT campo1, campo2 FROM tabela1 WHERE
      campo1 > 1000) AND (CALL
      (c:\usr\tricia\condicao.exe))',
'INSERT INTO tabela1 (campo1, campo2) VALUES (1,2);
      CALL (c:\usr\tricia\acao.doc)',
'1')
```

**Comando *Excluir_Regra***

**Objetivo**: Excluir uma regra ativa.

**Retorno**: Valor numérico. Assume os valores:

0 – Regra ativa excluída com sucesso

1 – Erro, regra ativa não existe

2 – Erro de sintaxe

**Sintaxe**:

```
Excluir_Regra(<nome_regra>)
```

**Exemplo 3.5**: Exclusão da regra ativa `R`, do exemplo anterior:

```
Excluir_Regra('R')
```

**Comando *Ativar_Regra***

**Objetivo**: Ativar uma regra ativa. Uma regra ativa só é processada se estiver ativada. Quando uma regra ativa é criada, seu estado inicial é desativada, permanecendo neste estado até que o ABD a ative por intermédio do comando `Ativar_Regra`.

**Retorno**: Valor numérico. Assume os valores:

0 – Regra ativa ativada com sucesso

1 – Erro, regra ativa não existe

2 – Erro de sintaxe

**Sintaxe**:

```
Ativar_Regra(<nome_regra>)
```

**Exemplo 3.6**: Ativando a regra ativa `R`:

```
Ativar_Regra('R')
```

**Comando *Desativar_Regra***

**Objetivo**: Desativar uma regra ativa.

**Retorno**: Valor numérico. Assume os valores:

0 – Regra ativa desativada com sucesso

1 – Erro, regra ativa não encontrada

2 – Erro de sintaxe

**Sintaxe**:

```
Desativar_Regra(<nome_regra>)
```

**Exemplo 3.7**: Desativando a regra ativa R:

```
Desativar_Regra('R')
```

### 3.2.3 Semântica de Execução de Regras do Engenho Ativo

Entende-se por semântica de execução de regras a maneira como regras ativas são processadas e como o processamento dessas regras deve interagir com o BD. Dentre as várias dimensões tratadas na definição de uma semântica de execução de regras [CW96], a especificação do Engenho Ativo se preocupou exclusivamente em definir prioridades de execução de regras. As demais dimensões da semântica de execução de regras como, modos de acoplamento e relacionamento com transações são "herdadas" do SGBD Base, isto é, do SGBD sobre o qual o Engenho Ativo é instalado, com exceção da granularidade que é sempre a nível de conjunto. A discussão de como essa herança é realizada é um dos temas do Capítulo 4, que trata da implementação do Engenho Ativo. A seguir são apresentadas as dimensões da semântica de execução de regras do Engenho Ativo.

### Modos de Acoplamento

O Engenho Ativo trata acoplamentos evento–condição (E-C) e condição–ação (C-A) como um espelho do que é oferecido pelo SGBD Base. Se no SGBD Base só forem permitidos os acoplamentos E-C e/ou C-A imediatos, o Engenho Ativo só permitirá os acoplamentos E-C e/ou C-A imediatos, implicitamente. Da mesma forma, se no SGBD Base só forem permitidos os acoplamentos E-C e/ou C-A retardados, o Engenho Ativo só permitirá os acoplamentos E-C e/ou C-A retardados, implicitamente. A herança vale também para as outras combinações de modos de acoplamento (por exemplo, E-C retardado e C-A imediato).

### Prioridade

O Engenho Ativo suporta um esquema de prioridades para resolver possíveis casos de conflitos de execução de regras [CW96]. Como foi visto no Capítulo 2, quando duas ou mais regras ativas são disparadas pelo mesmo evento, caracteriza-se um caso de conflito, surgindo então a necessidade de se estabelecer uma política de solução de conflitos. A solução de conflitos adotada pelo Engenho Ativo é baseada em prioridades. Desta forma, valores numéricos podem ser atribuídos às regras, visando definir uma ordem parcial de execução de um conjunto de regras disparadas pelo mesmo evento. O atributo prioridade pode assumir valores de 0 a 10, de acordo com a sintaxe:

```
<prioridade> → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
```

Por exemplo, sejam duas regras ativas `R1` e `R2` criadas com as prioridades 2 e 5, respectivamente, e disparadas pelo mesmo evento `E`. Quando o evento `E` ocorrer, a regra `R1` será processada `antes` da regra `R2` (quanto menor o número, maior a prioridade). Se `R1` e `R2` tivessem sido criadas com a mesma prioridade, o Engenho Ativo executaria as duas regras em uma ordem aleatória.

Ressaltamos que a dimensão prioridade é independente do SGBD Base, em outras palavras, a política de solução de conflitos é processada por inteiro pelo Engenho Ativo.

**Relacionamento com Transações**

O Engenho Ativo, sendo uma camada de software no topo de um SGBD, não tem meios de acessar o código fonte do SGBD no sentido de implementar sua própria política de relacionamento entre as regras ativas e as transações que acessam o BD. Para ilustrar, não é possível definir que uma regra ativa é uma transação independente da transação geradora do evento que a disparou, se esta funcionalidade não é suportada pelo SGBD Base. Resumindo, o Engenho Ativo "herda" do SGBD Base a sua dimensão relacionamento com transações.

**Granularidade**

Granularidade de regras [CW96] no Engenho Ativo só faz sentido em presença de eventos-SQL. Como foi descrito no Capítulo 2, o nível de granularidade define quando uma regra ativa será disparada por um evento-SQL, se antes ou depois de cada registro inserido/alterado/excluído em uma tabela por um comando SQL (nível instância), ou se antes ou depois do conjunto de registros inseridos/alterados/excluídos em uma tabela por um comando SQL (nível conjunto). A granularidade do Engenho Ativo é sempre a nível de conjunto pois, como os eventos-SQL são embutidos em procedimentos, os eventos que podem ser definidos são do tipo *antes do procedimento ser executado* ou *depois do procedimento ser executado*. Esta restrição é atenuada pelo fato de que os *triggers* normais dos SGBDs Base, com granularidade instância e conjunto, podem conviver normalmente com as regras ativas do Engenho Ativo.

Na sequência, é mostrado como as partes Evento, Condição e Ação se complementam para definir completamente uma regra ativa, através da linguagem de definição de regras ativas do Engenho Ativo.

## 3.3 Síntese Comparativa das Funcionalidades Ativas dos Principais SGBDs Comerciais com as do Engenho Ativo

A seguir, é apresentada uma tabela comparativa das funcionalidades ativas oferecidas pelos SGBD comerciais, cotejadas com as do Engenho Ativo.

| | Oracle 7 | Informix | Ingres | Sybase | Engenho Ativo |
|---|---|---|---|---|---|
| EVENTO | | | | | |
| Modificação de Dados | X | X | X | X | X |
| Recuperação de Dados | - | - | - | - | X |
| Definidos pela Aplicação | - | - | - | - | X |
| Disparo Antes | X | X | - | - | X |
| Disparo Depois | X | X | X | X | X |
| Evento Composto | X[8] | - | - | - | X |
| CONDIÇÃO | | | | | |
| Predicado SQL | - | X | X | - | X |
| Predicado SQL Restrito | X | - | - | - | - |
| Consultas ao BD | - | - | - | - | X |
| Procedimentos da aplicação | - | X | - | - | X |
| AÇÃO | | | | | |
| Modificação de Dados | X | X | - | X | X |
| Recuperação de Dados | X | - | - | - | X |
| Outros Comandos SQL | X | - | - | X | X |
| Procedimentos da Aplicação | X | X | X | X | X |
| SEMÂNTICA | | | | | |
| Prioridade | - | - | - | - | X |

**Tabela 3.1 - Síntese Comparativa das Funcionalidades Ativas dos Principais SGBDs Comerciais com as do Engenho Ativo**

Pela tabela acima, é fácil concluir que o Engenho Ativo contribui notavelmente para estender as funcionalidades ativas dos principais SGBDs comerciais.

[8] – Forma limitada de evento composto utilizando apenas o operador OR.

Algumas observações adicionais sobre a tabela comparativa devem ser feitas. Eventos-SQL são oferecidos indiretamente pelo Engenho Ativo, como foi descrito na Seção 3.2.1. É necessário criar um procedimento, e colocar o comando SQL (`INSERT` ou `SELECT`, por exemplo) no corpo do procedimento.

Quanto a eventos compostos, a tabela mostra que isto é possível no Oracle7; porém, vale salientar que é uma forma muito limitada de evento composto, em que o único operador lógico permitido é *OR*, e em que todas as ocorrências da composição devem ser operações sobre uma mesma tabela. Bem ao contrário do Oracle7, o Engenho Ativo suporta expressões complexas de eventos, envolvendo os operadores lógicos *AND*, *NOT* e *OR*, as ocorrências podendo ser operações em tabelas distintas, ou mesmo qualquer procedimento definido pelo usuário.

No que diz respeito à interface de manipulação de regras do Engenho Ativo, optou-se por uma linguagem de regras *não* SQL-like, pela dificuldade de encaixar funcionalidades como composição de eventos e prioridades dentro de uma sintaxe SQL.

# Projeto do Engenho Ativo

Neste capítulo, é apresentado o projeto do Engenho Ativo e são discutidos os detalhes da implementação do mesmo.

## 4.1 Arquitetura

De acordo com a arquitetura em camadas vista no Capítulo 2, o Engenho Ativo é uma camada de software implementada no topo de um SGBD Relacional (SGBD Base), fazendo uso de suas funcionalidades ativas para oferecer ao usuário um ambiente ativo como se este fizesse parte do SGBD Base. As regras ativas são mantidas pelo ABD, enquanto que o uso delas é totalmente transparente aos usuários comuns do SGBD Base.

A arquitetura do Engenho Ativo pode ser vista na Figura 4.1.

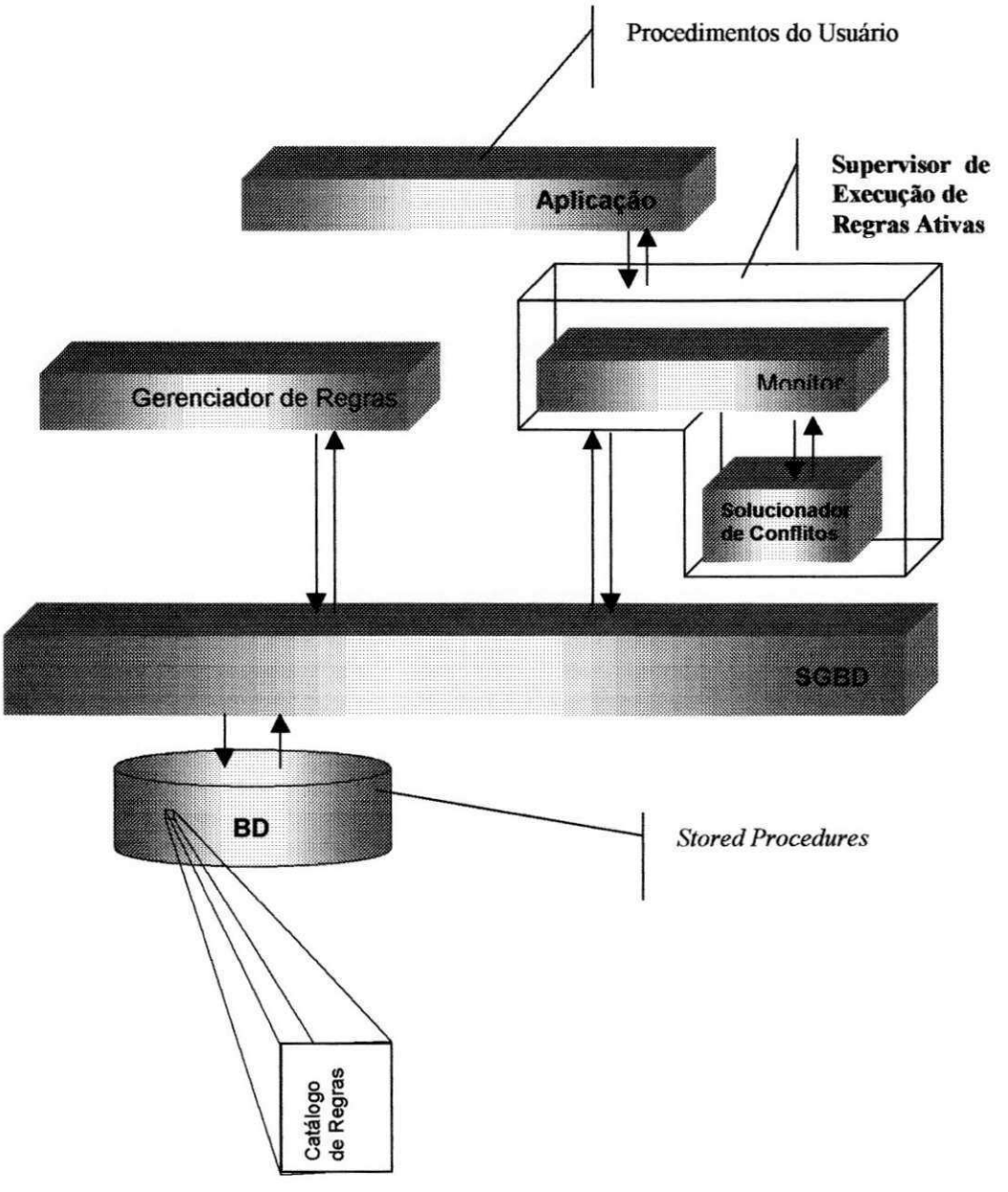



**Figura 4.1 – Arquitetura do Engenho Ativo**

Pode-se ver claramente que o Engenho Ativo se divide em dois grandes módulos, **Gerenciador de Regras** e **Supervisor de Execução de Regras Ativas**, este último com os sub-módulos **Monitor** e **Solucionador de Conflitos**. Na sequência, é descrito cada um dos módulos.

## 4.2 Gerenciador de Regras Ativas

Dentro da arquitetura do Engenho Ativo, o Gerenciador de Regras é o sub-módulo responsável pela interface do Engenho Ativo com o ABD. Ele suporta a linguagem de definição de regras descrita no Capítulo 3.

Na versão atual do Engenho Ativo, foi feita uma simplificação no que diz respeito à definição das condições e ações de regras ativas. Como procedimentos do usuário podem ser usados na definição da condição e da ação de uma regra ativa, o usuário deve embutir em uma função booleana a expressão que caracteriza uma condição. A condição será considerada se a função retornar o valor verdade. Para a ação é análogo: o usuário deve embutir em um procedimento o conjunto de comandos que formam a ação de uma regra, a ação propriamente dita para o Engenho Ativo sendo a chamada ao procedimento.

Retomando o exemplo de regra ativa do Capítulo 3:

```
Criar_Regra( 'R',
'DEPOIS ((INSERT_T1 OR DELETE_T2) AND P1)',
'(SELECT campo1, campo2 FROM tabela1 WHERE
      campo1 > 1000) AND (CALL
      (c:\usr\tricia\condicao.exe))',
'INSERT INTO tabela1 (campo1, campo2) VALUES (1,2);
 CALL (c:\usr\tricia\acao.doc)',
 '1')
```

Na versão atual do Engenho Ativo a mesma regra é criada da seguinte maneira:

```
Criar_Regra( 'R',
'DEPOIS ((INSERT_T1 OR DELETE_T2) AND P1)',
CONDIÇÃO,
AÇÃO,
'1')
```

A função booleana contendo a condição da regra poderia ser, numa sintaxe livre:

```
Function CONDIÇÃO : BOOLEAN
BEGIN
      IF (SELECT campo1, campo2 FROM tabela1 WHERE
            campo1 > 1000) AND CALL (c:\usr\tricia\condicao.exe)
            THEN RETURN TRUE
      ELSE
            RETURN FALSE
END
```

O procedimento contendo a ação da regra poderia ser, numa sintaxe livre:

```
Procedure AÇÃO
BEGIN
      INSERT INTO tabela1 (campo1, campo2) VALUES
                  (1,2)";
      CALL (c:\usr\tricia\acao.doc)
FIM
```

Aproveitando os cenários apresentados no Capítulo 3 para ilustrar o exemplo acima, tem-se:

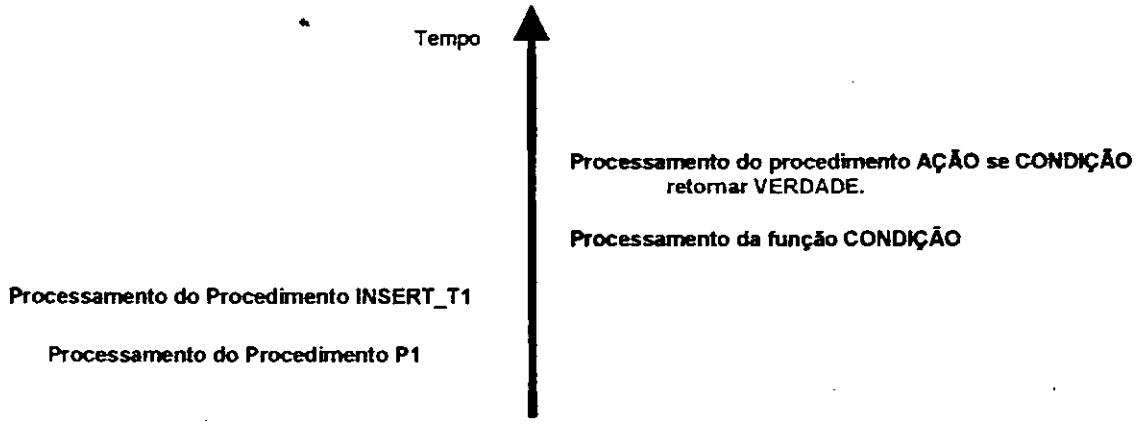



**Figura 4.2 – Processamento de uma Regra Ativa no Engenho Ativo**

O cenário acima assume que o processamento da regra ativa (condição, ação) é feito depois do processamento do evento. Note que a avaliação da condição passa a ser a chamada a função CONDIÇÃO e o retorno de um valor VERDADE significa que a

condição foi satisfeita e a ação deve ser executada. Da mesma forma o processamento da ação passa a ser a chamada ao procedimento AÇÃO.

O Gerenciador de Regras armazena e recupera as regras ativas de um **Catálogo de Regras**, que é uma tabela descrevendo regras, com os seguintes campos:

- Nome: Contém o nome da regra e deve ser único em todo o sistema, ou seja, não pode haver duas regras como o mesmo nome;
- Condição: Nome da função booleana que contém a definição da condição da regra;
- Ação: Nome do procedimento que contém a ação da regra;
- Evento: Conjunto de operações (ocorrências) e operadores lógicos que determina a expressão de eventos da regra;
- Atributo de Execução: Este campo indica se o processamento da regra ativa será efetuado ANTES ou DEPOIS do evento disparador, podendo receber os valores ANTES ou DEPOIS (*default* DEPOIS);
- *Status*: Indica se a regra esta ativada ou não (*default* não ativada);
- Prioridade: Indica a prioridade de execução da regra em relação às outras existentes no sistema (*default* prioridade máxima);

Agora será descrita a lógica de cada comando da interface de manutenção de regras ativas definida no Capítulo 3.

### 4.2.1 Lógica dos Comandos da Linguagem de Manutenção de Regras Ativas

#### Comando `Criar_Regra`

Quando um usuário submete seu pedido de catalogação de uma nova regra ativa, o Gerenciador de Regras inicialmente verifica no Catálogo de Regras a existência de uma regra ativa catalogada com o mesmo nome. Se existir, é retornado um erro e a regra não é catalogada, senão a nova regra ativa é cadastrada no Catálogo de Regras.

Os eventos definidos pelo usuário envolvem chamadas a procedimentos armazenados no BD (*stored procedures*) ou a rotinas externas (escritas em uma

linguagem qualquer e armazenadas fora do BD). Tanto os procedimentos armazenados como as rotinas externas devem ter seu código fonte preparado para caracterizar um evento, ou seja, para avisar ao Engenho Ativo da sua chamada. Os eventos-SQL se comportam de forma similar pois eles nada mais são do que comandos SQL embutidos em procedimentos armazenados ou rotinas externas. Porém, o Engenho Ativo permite criar regras e deixá-las desativadas, isto é, mesmo que aconteça o evento disparador daquela regra ela não será processada. Com isso, uma regra, cuja expressão de eventos envolva um procedimento ainda não preparado para caracterizar um evento, pode ser criada e ativada só após os procedimentos envolvidos na expressão de eventos estarem aptos a caracterizar eventos.

A preparação de um procedimento, para que este passe a caracterizar um evento, consiste em colocar uma chamada ao Monitor do Engenho Ativo dentro do código fonte do procedimento. O ponto onde deve ser colocada a chamada ao Monitor depende se o evento é antes ou depois da chamada ao procedimento. Se for antes, então a chamada ao Monitor deve ser feita no início do procedimento, isto é, antes de qualquer outro comando do procedimento, devendo o procedimento propriamente dito entrar em estado de espera até que o Monitor termine de processar a regra ativa concernente. Se, ao contrário, o evento é depois da chamada ao procedimento, a chamada ao Monitor deve ser colocada como o último comando do procedimento.

Considere agora um evento composto. Seja, por exemplo, o evento `ANTES ((INSERT_T1 OR DELETE_T2) AND P1)`. Neste caso, a regra ativa associada será disparada antes do último evento simples ocorrido que faça da expressão de eventos uma expressão verdadeira. Uma das possibilidades é: quando o procedimento `P1` é chamado, a expressão de eventos ainda não é satisfeita; então `P1` entra em estado de espera até que seja chamado o procedimento `INSERT_T1`, quando então é disparada a regra ativa associada. Quando terminar o processamento da regra ativa, são verdadeiramente processados.

Mais detalhadamente, a expressão de eventos acima assume a forma `((FALSO OR FALSO) AND FALSO)`, inicialmente; na medida em que os eventos da expressão de eventos vão acontecendo, os valores `FALSO` vão sendo substituídos por `VERDADEIRO`. No

nosso exemplo, a expressão de eventos muda para a forma `((VERDADEIRO OR FALSO) AND FALSO)` e depois para `((VERDADEIRO OR FALSO) AND VERDADEIRO)`.

**Comando `Excluir_Regra`**

O comando `Excluir_Regra` tem como objetivo excluir regras ativas do Catálogo de Regras. O Gerenciador de Regras verifica se a regra ativa existe e remove a sua entrada no Catálogo de Regras. A remoção da chamada ao Monitor de dentro dos procedimentos que caracterizam eventos, só é feita se àquele procedimento-evento só fizer parte da expressão de eventos da regra ativa que está sendo removida.

**Comando `Ativar_Regra`**

O Gerenciador de Regras, quando cria uma regra ativa, deixa-a desativada até que o ABD mude o seu estado explicitamente pela chamada ao comando `Ativar_Regra`. Quando o comando `Ativar_Regra` for executado, o Gerenciador de Regras procura a regra ativa no Catálogo de Regras e muda o seu estado para *Ativa*.

**Comando `Desativar_Regra`**

O Engenho Ativo oferece a opção de desativar uma regra ativa sem que para isto o usuário precise excluí-la do Catálogo de Regras. Esta opção é oferecida sob a forma do comando `Desativar_Regra`.

Esta característica é importante quando o Módulo Ativo precisa sofrer alguma manutenção. O ABD pode simplesmente desativar as regras ativas criadas e depois da manutenção concluída, reativá-las.

### 4.2.2 Interface Usuário – Gerenciador de Regras

Esta seção é dedicada à interface ABD – Gerenciador de Regras. Inicialmente, é apresentada a estrutura do Gerenciador de Regras, depois como cada atividade de manipulação de regras é realizada, e finalizando, alguns detalhes da lógica de implementação.

**Estrutura do Gerenciador de Regras**

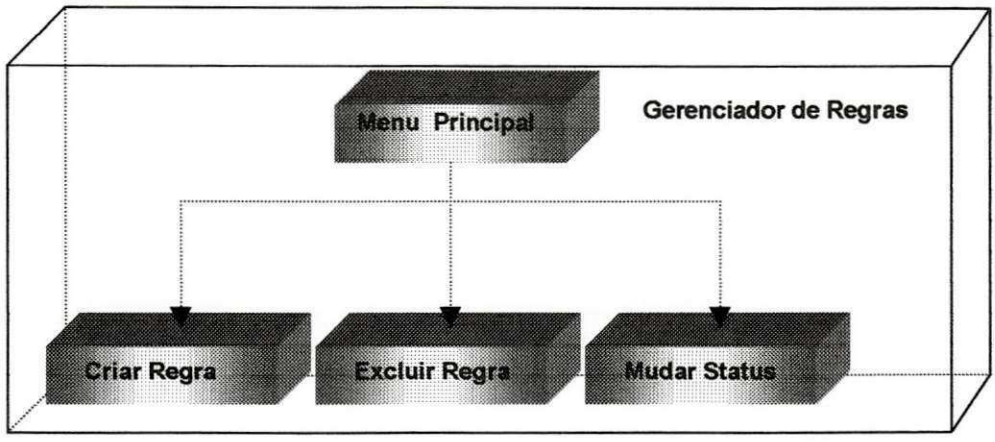



**Figura 4.3 - Estrutura do Gerenciador de Regras**

Quando o Gerenciador de Regras é executado, é apresentada ao usuário a tela inicial, composta por um menu de opções que caracteriza os comandos de manipulação de regras ativas oferecidos pelo Engenho Ativo. Além do menu, a tela inicial fornece uma **"barra de tarefas"**, que possibilita ao usuário acesso mais rápido aos comandos de manipulação de regras (Figura 4.4).

Os comandos foram implementados sob a forma de janelas que recebem os parâmetros em campos editáveis e botões de múltipla escolha. Cada janela tem sua funcionalidade bem definida, ou seja, cada janela executa apenas uma atividade.


**Figura 4.4 – Menu Principal do Gerenciador de Regras**


Abre tela de criação de regras ativas

Abre tela de exclusão de regras ativas

Abre tela de mudança de status de regras ativas

**Figura 4.5 – Funções dos botões presentes na Barra de Tarefas**

A Figura 4.5 descreve a função de cada botão presente na Barra de Tarefas.

**Janelas de Manipulação de Regras Ativas**

**Janela *Criar***

Esta janela permite ao usuário fornecer os parâmetros para o comando Criar_Regra, que tem como objetivo a inclusão de uma nova regra ativa no Catálogo de Regras.

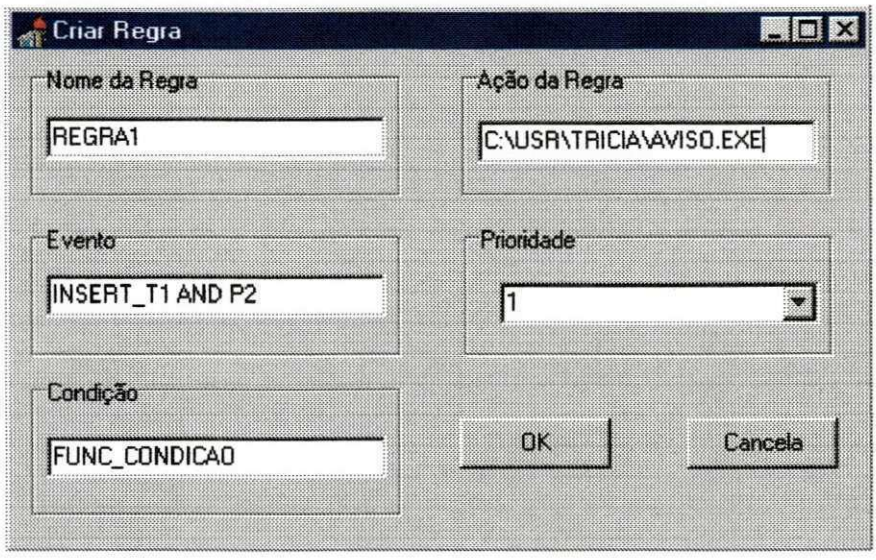



**Figura 4.6 - Janela Criar Regra**

Na janela Criar, o usuário deve informar todos os campos necessários à criação de uma regra ativa. É nesta janela que se define a expressão de eventos que irá disparar a regra, o nome da função como condição da regra, o nome do procedimento como ação da regra, granularidade e a prioridade de execução da regra em caso de conflito. Note que o atributo *status* não é definido diretamente pelo usuário no momento da criação da regra, mas sim, internamente pelo Engenho Ativo. Este atributo é inicializado com o seguinte valor:

| Atributo | Valor Inicial |
|---|---|
| Status | 'D' ( desativada) |

**Tabela 4.1 - Valores iniciais de atributos de regras**

As regras ativas são criadas com o atributo *status* igual a 'D' (desativada), conforme explicado anteriormente. No momento devido, o ABD pode ativar regras explicitamente, em janela específica.

Um detalhe importante em relação à condição de uma regra ativa é que, se no momento da criação o campo **Condição** ficar em branco, isto indica que a condição da regra sendo criada é sempre verdadeira. Mais precisamente, quando do evento disparador, a ação da regra será executada sem nenhuma restrição.

Após o usuário clicar no botão **OK**, é feita uma pesquisa no Catálogo de Regras para verificar se existe uma regra ativa com o mesmo nome da regra que está sendo criada. Se existir, a regra não será criada, e o Engenho Ativo informará ao ABD que já existe uma regra ativa catalogada com o mesmo nome.

**Janela *Excluir***

A janela Excluir permite ao usuário fornecer o nome de uma regra ativa para o comando Excluir_Regra, que tem como objetivo a exclusão de uma regra do Catálogo de Regras. Esta janela é apresentada na Figura 4.7, disponibilizando ao usuário um campo denominado **Nome da Regra**, onde ele poderá digitar o nome da regra a ser excluída, ou escolher, através de uma lista de todas as regras ativas (*browsing*), qual regra deve ser excluída.

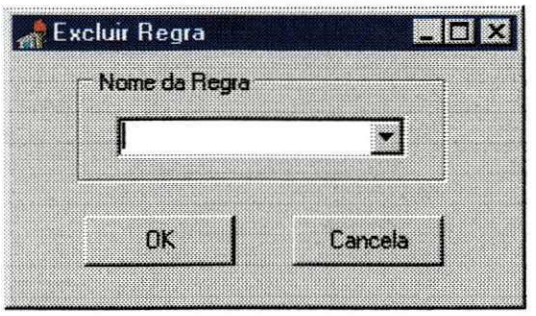



**Figura 4.7 - Janela Excluir Regra**

Após o usuário clicar no botão **OK,** é feita uma pesquisa no Catálogo de Regras para verificar se a regra a ser excluída está catalogada. Se a regra existir, ela será excluída. Caso contrário, o Engenho Ativo retornará ao usuário uma mensagem de erro, informando que não encontrou a regra.

### Janela *Status* de Regra

Na janela *Status* de Regra, o usuário pode ativar ou desativar uma regra ativa, ou seja, habilitar uma regra para ser processada ou não.

Como foi visto, regras ativas são criadas com o *status* desativada, significando que, mesmo em presença do evento disparador da regra, esta não será processada pelo Monitor. Portanto, a atividade de mudança de *status* deve ser executada ao menos uma vez para cada regra, quando ela é ativada a primeira vez.

A janela *Status* de Regra (Figura 4.8) pode ser chamada através da opção do menu **Regra / Status** ou da barra de tarefas. Esta janela possui um campo denominado **Nome da Regra,** onde da mesma forma que na atividade **Excluir Regra,** o usuário pode digitar o nome ou escolher a regra ativa da qual ele deseja mudar o status (ativada ou desativada), através de uma lista com todas as regras existentes no sistema. Além do campo **Nome da Regra**, esta janela disponibiliza dois campos denominados **Ativada** e **Desativada,** mutuamente exclusivos.


**Figura 4.8 - Janela Mudar o *Status* da Regra**

Considere que uma regra tem o *status* desativada, e que o usuário deseja ativá-la. Quando o usuário determina o nome da regra, o Engenho Ativo faz uma pesquisa no Catálogo de Regras. Se a regra ativa desejada for encontrada, a mudança de *status* é processada, caso contrário, a aplicação retorna ao usuário uma mensagem informando que a regra não está catalogada.

Os algoritmos em pseudo-código do Gerenciador de Regras encontram-se no Anexo 1.

## 4.3 Supervisor de Execução de Regras Ativas

### 4.3.1 Visão Geral

O Monitor e o Solucionador de Conflitos são os módulos responsáveis pela supervisão da execução das regras ativas criadas e catalogadas pelo Gerenciador de Regras. O Monitor recebe as notificações da ocorrência de um evento e é responsável pela busca da(s) regra(s) ativa(s) relacionadas ao evento ocorrido, pela avaliação da expressão de eventos e da condição das regras ativas selecionadas, e pela execução de suas respectivas ações.

Quando ocorre um evento, o Monitor é avisado explicitamente pelos procedimentos que caracterizam os eventos, como foi visto. Ele se encarrega de pesquisar no Catálogo de Regras as regras ativas relacionadas ao evento. Caso a

expressão de eventos seja avaliada como verdadeira, o Monitor insere cada regra ativa relacionada na tabela de regras candidatas à execução; no final da busca no Catálogo de Regras, o Monitor ordena a execução das regras ativas na tabela de regras candidatas a execução, de acordo com a política de solução de conflitos adotada.

A execução de uma regra consiste na consideração de sua condição e, esta sendo verdadeira, a ação da regra é executada.

Seja o seguinte cenário: Uma regra ativa `R` é criada com o evento `ANTES (INSERT_T1 AND P1)`. Quando o procedimento `P1` for chamado, a primeira linha executada em `P1` é uma chamada ao Monitor passando como parâmetro o evento `ANTES P1`. Observe que `P1` fica esperando a volta da chamada ao Monitor para poder continuar o processamento de seus comandos. O Monitor então procura no Catálogo de Regras as expressões de eventos que possuam no seu interior o evento simples ocorrido, no caso `P1`. Ele deve achar a expressão de eventos da regra ativa `R`. Porém, apenas com `P1`, a expressão não se torna verdadeira, e a regra `R` não é ainda processada. Apenas na presença também do evento `ANTES INSERT_T1` é que a regra será processada.

Examinando agora a existência de conflitos. Se existisse uma regra ativa `R1` disparada pelo evento `ANTES INSERT_T1`, então as regras `R` e `R1` seriam disparadas ao mesmo tempo, ou `R1` primeiro, ou `R` primeiro, conforme as prioridades de `R` e `R1` fossem iguais, ou a prioridade de `R1` fosse maior, ou a prioridade de `R` fosse maior, respectivamente.

### 4.3.2 Arquitetura do Monitor

A arquitetura do Monitor é apresentada na Figura 4.9. Como o módulo Solucionador de Conflitos é muito simples, ele foi embutido no Monitor.

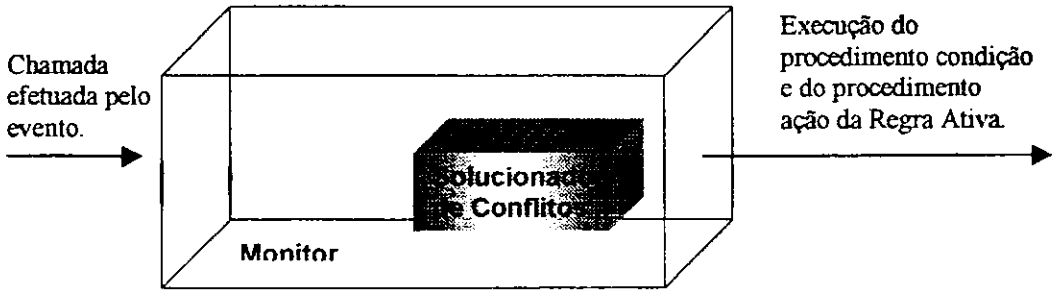



**Figura 4.9 - Estrutura do Monitor**

O Monitor manipula diretamente duas tabelas, criadas no momento de instalação do Engenho Ativo. São elas o Catálogo de Regras e a Tabela de Regras Candidatas a Execução, que serve de interface entre o Monitor e o Solucionador de Conflitos. Esta última tabela não é mostradas na arquitetura da Figura 4.9 mas está descritas no Anexo 3, juntamente com o Catálogo de Regras.

A tabela Regras Candidatas a Execução é utilizada para armazenar as regras candidatas. Ela serve de entrada para o Solucionador de Conflitos que a usa para ordenar a execução das regras. Possui apenas dois campos:

- nome_regra: Nome da regra ativa. Do tipo cadeia de caracteres;
- prioridade: Indica a prioridade de execução da regra. Assume valores de 0 a 10. Do tipo numérico.

### 4.3.3 A Implementação do Monitor

O Monitor se apresenta como uma rotina executável que fica residente na memória, à espera de algum evento. Quando ele é chamado pelo ABD, é mostrada a seguinte janela:


**Figura 4.10 – Interface do Monitor**

Como se pode notar, a interface do Monitor é extremamente simples, apresentando um único botão com a finalidade de terminar a execução. Porém, é de extrema importância que, para finalizar a execução do Monitor, se use o botão *Finalizar*, pois desta forma ele se certifica de que nenhuma regra está sendo processada no momento, ou então, se existir alguma regra em processamento, o Monitor espera o término do processamento de todas as regras e depois finaliza sua execução.

Como se sabe, o Monitor é notificado da ocorrência de um evento simples por uma chamada feita de dentro de um procedimento do usuário. A notificação se dá através de um mecanismo de passagem de mensagens, cuja sintaxe é apresentada a seguir:

```
<mensagem> ::= (<momento>,<procedimento>)
<momento> ::= ANTES | DEPOIS
```

A chamada ao Monitor depende do SGBD Base sendo usado. A título de ilustração, no Oracle7, as *stored procedures* só podem ser escritas em PL/SQL, com a limitação de não permitir chamar rotinas externas de forma direta. Detalhes serão vistos no próximo capítulo, que trata da implementação do Engenho Ativo no topo do Oracle7.

Quando o Monitor é chamado, ele procura no Catálogo de Regras as regras ativas cujas expressões de eventos contemplem o evento simples da mensagem recebida. A expressão de eventos é então avaliada e, se verdadeira, e se as regras ativas

relacionadas estiverem ativadas, as regras são incluídas na Tabela de Regras Candidatas à Execução, para a detecção e solução de conflitos por execução das regras.

Nas duas sub-seções seguintes, são explicados em mais detalhes como o Monitor avalia uma expressão de eventos, e como ele detecta e soluciona conflitos por execução de regras.

**Avaliação da Expressão de Eventos**

Na discussão que se segue, e apenas para facilitá-la, será omitida a cláusula {ANTES | DEPOIS} da definição de um evento. O leitor poderá verificar que esta simplificação não traz nenhum prejuízo ao essencial da questão.

Para ilustrar, será utilizado o seguinte exemplo de regra ativa:

```
Criar_Regra('Z',
      '(INSERT_T1 AND P1)',
      'CONDIÇÃO','AÇÃO','1')
```

Uma expressão de eventos, como a da regra Z, é armazenada como uma cadeia de caracteres em que, logo após cada evento simples, é colocado um caracter de controle para identificar se aquele evento simples já ocorreu ou não, ou que ele foi sinalizado ou não. Isto é feito para facilitar o tratamento de eventos compostos. A expressão de eventos armazenada no Catálogo de Regras por ocasião da criação de Z é a seguinte:

```
INSERT_T1-  AND  P1-
```

Os sinais '-' indicam que os eventos simples respectivos ainda não foram sinalizados. A cada operando <objeto>- é atribuído o valor FALSO. Daí a expressão acima ser assim avaliada inicialmente:

```
FALSO AND FALSO   ≡   FALSO
```

Em presença da chamada à rotina INSERT_T1, a ocorrência de um evento é identificada pela chamada ao Monitor, que é feita dentro do procedimento INSERT_T1. O Monitor procura então no Catálogo de Regras as regras ativas que possuem na sua expressão de eventos o evento simples INSERT_T1. Note que, sempre que ocorrer um evento, o Monitor deve encontrar pelo menos uma regra ativa associada a esse evento no Catálogo de Regras. Se isto não acontecer, então terá sido caracterizada uma situação de erro.

No nosso exemplo, o Monitor acha o evento simples INSERT_T1 como operando da expressão de eventos associada à regra ativa Z. O Monitor substitui o sinal '-' por '+', indicando assim que o evento INSERT_T1 dentro da expressão de eventos associada a Z foi sinalizado (valor VERDADE associado ao operando). O leitor pode facilmente perceber que, neste ponto, a expressão inteira ainda não foi sinalizada, ou não pôde ainda receber o valor VERDADE. Ela o receberá em presença do evento P1, quando então tem-se:

```
        INSERT_T1+  AND P1+
VERDADE AND VERDADE   ≡   VERDADE
```

O próximo passo é a organização da Tabela de Regras Candidatas à Execução, quando então os conflitos por execução de regras são detectados e solucionados. É importante saber que neste ponto, na expressão de eventos todos os sinais '+' voltam a ser '-', para evitar que as regras associadas à expressão sejam novamente selecionadas para execução.

**Detecção e Solução de Conflitos**

A organização da Tabela de Regras Candidatas à Execução, por ordem de prioridade de execução, fica a cargo do sub-módulo Solucionador de Conflitos do Monitor. Suponha que, para a expressão de eventos do exemplo da seção anterior, exista uma outra regra ativa X, além da regra Z do exemplo. Tem-se então o seguinte *snapshot* da Tabela de Regras Candidatas à Execução:

| Regra | Prioridade |
|---|---|
| X | 2 |
| Z | 1 |

Note então que o conflito entre as regras x e z é resolvido fazendo z executar primeiro, porque sua prioridade é maior que a de z (lembrar que quanto menor é o número, maior é a prioridade).

O Anexo 2 contém a descrição do algoritmo em pseudo-código o Monitor.

## 4.4 Fluxo de Mensagens do Engenho Ativo

Esta seção resume o fluxo de mensagens entre processos do Engenho Ativo, com o significado de cada uma delas. Na Figura 4.11, um tipo i de mensagem é representado por uma seta etiquetada de Mi.

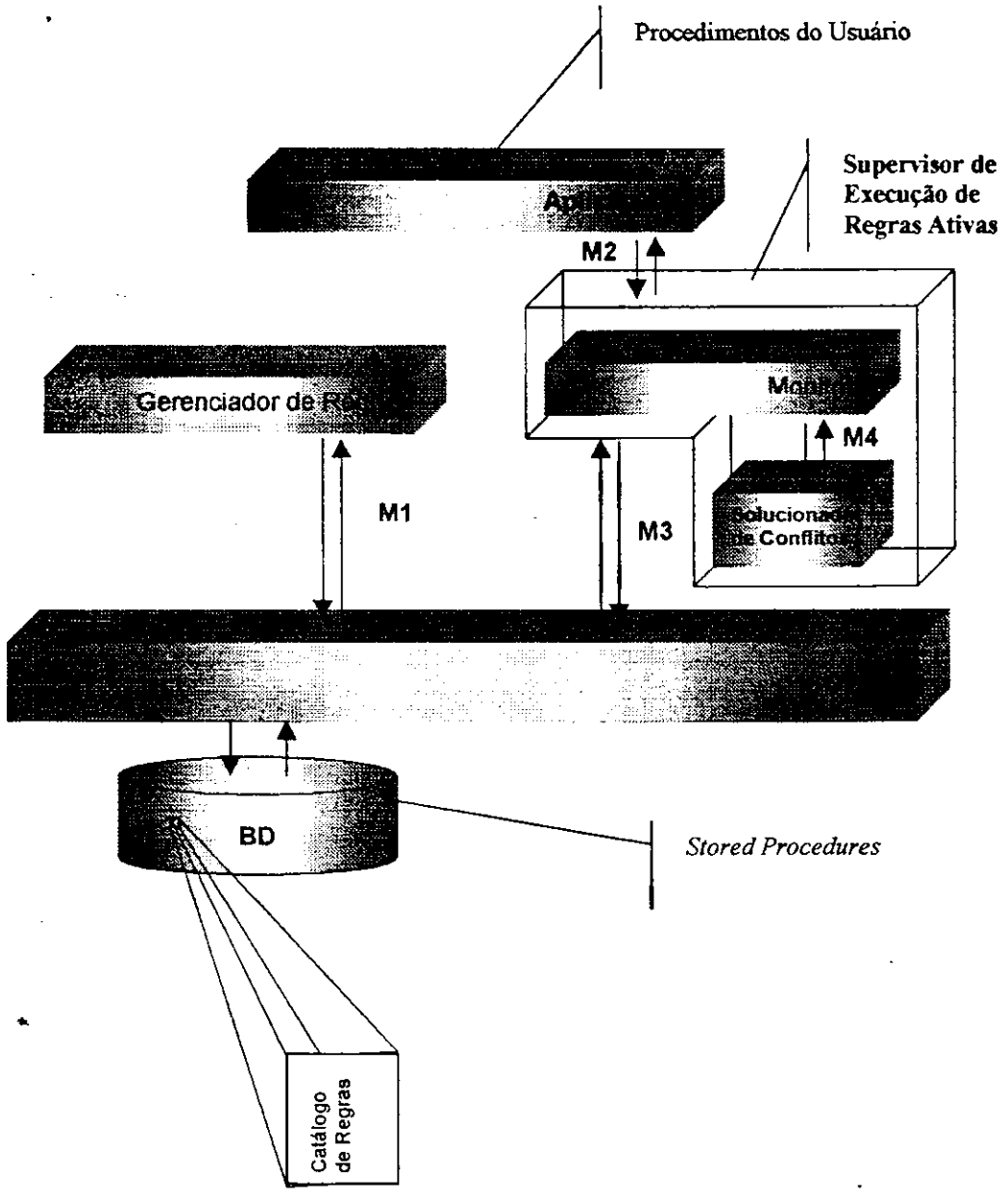



**Figura 4.11 – Fluxo de Mensagens**

M1: comandos SQL sobre o SGBD Base, efetuados pelo Gerenciador de Regras como, por exemplo, inserções de registros na tabela Catálogo de Regras quando da criação de uma regra ativa;

M2: o fluxo que desce representa uma chamada ao Monitor por um evento-procedimento-usuário do tipo rotina externa ao SGBD Base. O fluxo que

sobe do Monitor para a Aplicação representa a chamada à condição ou à ação de uma regra ativa quando essas são rotinas externas;

**M3** : o fluxo que sobe representa uma chamada ao Monitor por eventos-SQL, ou eventos-procedimento-usuário do tipo *stored procedures*. O fluxo que desce do Monitor para o SGBD representa a chamada à condição ou à ação de uma regra ativa quando essas são *stored procedures*;

**M4** : caracteriza a entrega das Regras Ativas candidatas a execução para que o Solucionador de Conflitos ordene a execução.

# O Engenho Ativo no Topo do Oracle7

O primeiro protótipo do Engenho Ativo foi implementado para dotar o SGBD Oracle7 de novas funcionalidades ativas, principalmente eventos definidos pelo usuário e eventos compostos.

O protótipo construído no topo do Oracle7 serviu também para validar a especificação do Engenho Ativo, a validação consistindo na transformação de fato de um SGBD semi-ativo em ativo, e na verificação do desempenho do sistema proposto.

Entretanto, a implementação do protótipo não seguiu à risca o que foi proposto na especificação. As razões para isto foram, na maior parte, as limitações impostas pela

linguagem de desenvolvimento do Oracle 7.3, a PL/SQL. A plataforma utilizada foi o SGBD Oracle 7.3 for Workgroups para o sistema operacional Windows NT 4.0.

Os módulos do Engenho Ativo clientes do Oracle7 foram desenvolvidos em Delphi 3.0. A aplicação de teste do Engenho Ativo também foi desenvolvida em Delphi 3.0.

No Oracle7, as regras de integridade do banco de dados devem ser implementadas na linguagem proprietária PL/SQL, através de ações de *triggers*, funções, procedimentos ("*stored procedures*") e pacotes ("*packages*"). Infelizmente, a linguagem PL/SQL não permite chamadas a rotinas externas ao banco de dados, como é o caso do módulo Monitor do Engenho Ativo.

Devido à limitação da linguagem PL/SQL de não fazer chamadas a rotinas externas, a solução encontrada foi substituir a chamada direta ao Monitor, quando da ocorrência de um evento, pela inserção de registros representando eventos em uma tabela denominada de Tabela Ativa. Considere, por exemplo, o procedimento-evento `INSERT_T1`, que encapsula a inclusão de um registro na tabela `T1`: segundo a especificação, em `INSERT_T1` deve ser colocado, no início ou no fim do procedimento conforme o evento seja definido como *antes* ou *depois* da execução do procedimento, uma chamada ao Monitor avisando-o da ocorrência do evento. No Oracle7 isto não é possível, porque a rotina Monitor não é escrita em PL/SQL. Para contornar o problema, a chamada ao Monitor é feita de forma indireta: primeiro, o procedimento-evento insere, numa tabela-Oracle chamada de Tabela Ativa, um registro representando a ocorrência do evento; depois, e em tempos determinados, o Monitor consulta a Tabela Ativa para descobrir a ocorrência do evento. Como esta consulta é feita de maneira assíncrona em relação à ocorrência do evento (a inserção do registro na Tabela Ativa caracterizando a ocorrência de um evento), o protótipo do Engenho Ativo para Oracle7 não consegue tratar eventos do tipo *antes* da execução de um procedimento.

Visualizando melhor tem-se:

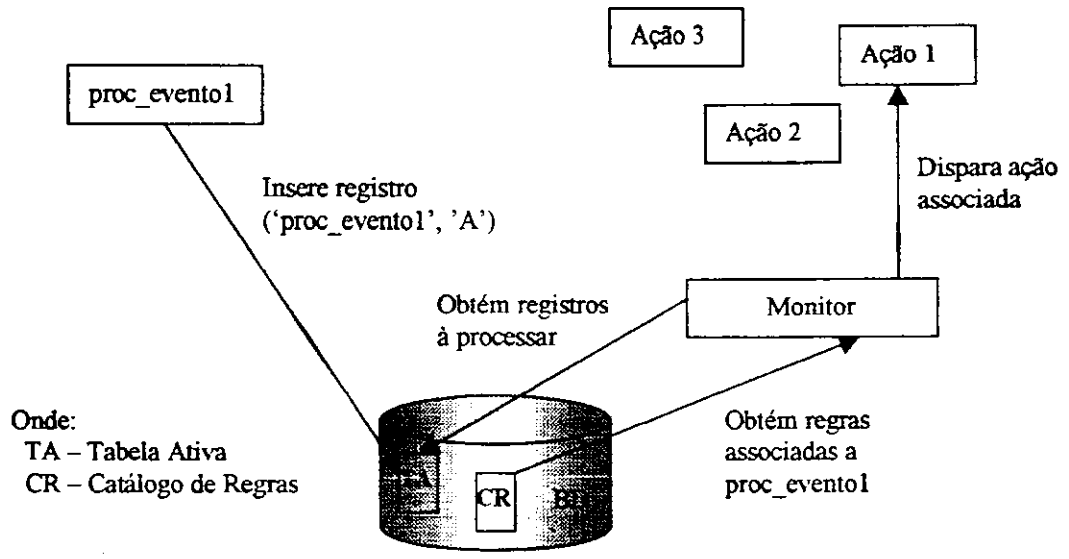



**Figura 5.1 – O Engenho Ativo no Oracle7**

Detalhando esta sequência de atividades:

- criação de uma tabela denominada de Tabela Ativa, que somente contém registros representando eventos ocorridos;
- quando uma regra ativa é criada via a interface de definição de regras do Engenho Ativo, em cada procedimento-evento da expressão de eventos da regra é colocada uma chamada a um procedimento que inclui na Tabela Ativa um registro representando o evento processado;
- de tempo em tempo, o Monitor consulta a Tabela Ativa para ver se existe algum registro novo na mesma;
- havendo registro novo na Tabela Ativa, o Monitor desencadeia a procura das regras ativas associadas aos novos eventos. Não havendo registros novos, o Monitor espera um determinado período de tempo e volta a consultar a Tabela Ativa.

A chamada ao procedimento que inclui os registros na Tabela Ativa deve ser colocada dentro do procedimento-evento pelo usuário, nesta versão. Ou seja, o usuário deve preparar o procedimento manualmente antes de criar a Regra Ativa associada.

As ações das regras podem ser representadas por *stored procedures* e por chamadas a rotinas executáveis, como foi especificado. Entretanto, a chamada a outros tipos de arquivos também caracterizam ações, ou seja, pode-se fazer referência a qualquer tipo de arquivo: se for uma rotina executável, o Monitor executa esta rotina; quando for um arquivo ".doc", o Monitor abre o arquivo no programa associado, no caso o processador de textos *Word*, e assim por diante.

## 5.1 Tabela Ativa

A Tabela Ativa é o meio de armazenamento de todos os eventos ocorridos. Tem a seguinte simples estrutura:

- Procedimento-evento: o conjunto de valores desta coluna consiste das chamadas aos procedimentos geradores de evento. Na realidade, é armazenado o nome do procedimento ou o nome que foi dado a ele quando da especificação da expressão de eventos no momento da criação da regra;
- status-registro: indica se o registro já foi processado ou não pelo Monitor. Ou seja, quando o Monitor acessa a Tabela Ativa, ele procura os novos registros, aqueles que caracterizam os eventos que ocorreram e que ainda não foram tratados. À medida que estes registros vão sendo tratados pelo Monitor, ele vai deixando uma marca para evitar que um registro seja processado mais de uma vez. Nesta coluna é armazenada o estado atual do registro assumindo os valores: 'A' para "A Processar" ou 'P' para "Processado".

No Anexo 3 é apresentado o *script* de criação da Tabela Ativa no Oracle7.

Nas seções seguintes, são mostrados os detalhes de como o Monitor do Engenho Ativo utiliza a Tabela Ativa para controlar as execuções das regras definidas por meio da interface de definição de regras do Engenho Ativo.

## 5.2 Implementação do Gerenciador de Regras no Oracle7

Nesta seção, são descritos alguns detalhes da implementação, no Oracle7, das atividades de manipulação de regras ativas, por meio da interface de manipulação de

regras ativas, conforme a especificação do Engenho Ativo. As atividades de manipulação de regras compreendem: criação de regras, exclusão de regras, mudança de *status* de regras e atribuição de prioridades a regras.

### 5.2.1 Criação de Regras

Esta atividade é realizada toda vez que o usuário deseja definir uma nova regra ativa. Antes de entrar em detalhes sobre a atividade de criação de regras é interessante relembrar a estrutura do Catálogo de Regras apresentada na seção 4.2 do capítulo anterior e verificar que a diferença entre a estrutura lá especificada e a implementada diz respeito à retirada do campo Atributo de Execução, devido à limitação de só aceitar eventos do tipo *depois* na versão Oracle7.

- Nome: Nome da regra. Único em todo o sistema;
- Condição: Nome da função booleana que contém a definição da condição da regra;
- Ação: Nome do procedimento que contém a ação da regra;
- Evento: Conjunto de operações (ocorrências) e operadores lógicos que determinam a expressão de eventos da regra;
- *Status*: Indica se a regra esta ativada ou não (*default* não ativada);
- Prioridade: Indica a prioridade de execução da regra em relação às outras existentes no sistema (*default* prioridade máxima);

Pode-se dividir a atividade de criação de regras em quatro etapas distintas:

- Etapa de Atribuição: nesta etapa, o usuário informa ao Engenho Ativo as características da nova regra ativa, quais sejam, o nome da regra, a expressão de eventos, a condição, a ação e o valor da prioridade da regra. A interface foi um pouco modificada pela ausência de eventos do tipo *antes*. A janela de criação de regra ficou a seguinte:


**Figura 5.2 – Janela de Criação de Regras do Engenho Ativo versão para Oracle7**

- Etapa de Pesquisa: esta etapa se inicia no momento em que o usuário "clica" com o *mouse* sobre o botão **"OK"** da janela **Criar Regra** (Figura 5.2). A partir desse momento, o Engenho Ativo, através do seu Gerenciador de Regras, faz uma pesquisa no Catálogo de Regras (ver *script* de criação do Catálogo de Regras para o Oracle7 no Anexo3) para ver se já existe uma regra com o nome dado. Em caso afirmativo, a nova regra não é incluída no Catálogo de Regras e o Gerenciador de Regras retorna ao usuário a mensagem "Já existe uma regra com o mesmo nome " fechando a janela **Criar Regra**. Se porém não existir no Catálogo de Regras uma regra com o nome fornecido, as próximas etapas são executadas;

- Etapa de Sinalização: fundamental para o processamento dos eventos compostos como foi visto no capítulo anterior. Consiste em percorrer a expressão de eventos da regra e, para cada evento simples, inserir um sinal de *evento não sinalizado*("-"). Por exemplo, a etapa de sinalização transforma a expressão de eventos acima, `INSERT_T1 AND P2`, na expressão `INSERT_T1-ANDP2-`, antes de armazená-la no Catálogo de Regras;

- Etapa de Catalogação: responsável pela inclusão de um novo registro no Catálogo de Regras. O Gerenciador de Regras, a partir das informações da

regra obtidas na janela de criação de regras monta o comando SQL para inserir uma nova regra no Catálogo de Regras[8]:

```
INSERT INTO Catalogo_Regras (
   cr_nome_regra, cr_condicao, cr_acao,
   cr_status, cr_prioridade, cr_expr_eventos)
VALUES ('REGRA1', 'FUNC_CONDICAO', 'C:\USR\TRICIA\AVISO.EXE',
   'D', 1, 'INSERT_T1-ANDP2-' );
```

O prefixo `cr` antes do nome dos campos da tabela faz referência ao nome da tabela, `Catalogo_Regras`. Após esta etapa a tarefa de criação de uma regra ativa está finalizada.

### 5.2.2 Exclusão de Regras

Esta atividade tem o objetivo de excluir uma Regra Ativa. A interface utilizada é idêntica a apresentada no capítulo anterior como pode-se ver na Figura 5.3:

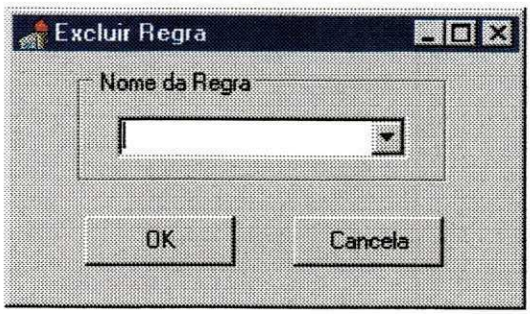



**Figura 5.3 – Janela de Exclusão de Regras do Engenho Ativo versão para Oracle7**

A atividade de exclusão de regras é realizada em apenas uma etapa:

- Etapa de Pesquisa e Exclusão: inicia-se após o usuário clicar sobre o botão **"OK"**. O objetivo é encontrar uma regra com o nome desejado. Se esta etapa não encontrar nenhuma Regra Ativa no Catálogo de Regras com o nome fornecido, a "etapa de exclusão" não será executada. Para excluir a regra ativa criada na seção anterior, o usuário deve entrar com REGRA1 no campo

[8] – Note que os nomes dos campos utilizados nos comandos são os nomes que foram usados no *script* de criação do catálogo de regras apresentado no Anexo 3.

Nome da Regra da janela de exclusão de regras apresentada acima e clicar sobre o botão OK. Internamente o gerenciador de regras monta o seguinte comando para a busca da regra dentro do Catálogo de Regras:

```
DELETE Catalogo_Regras
      WHERE cr_nome_regra = 'REGRA1';
```

Note que a consulta deverá excluir apenas um registro pois a atividade de criação de regras só permite criar uma regra com um determinado nome.

### 5.2.3 Mudança de *Status* de Regras

Esta atividade é realizada toda vez que o usuário deseja modificar o *status* de uma Regra Ativa, ou seja, ativar ou desativar uma regra ativa catalogada. A interface utilizada também é idêntica a que foi apresentada no capítulo anterior:


**Figura 5.4 – Janela de Mudança de *Status* de Regras do Engenho Ativo versão para Oracle7**

Esta atividade foi dividida em duas etapas:

- Etapa de Atribuição: nesta etapa, o usuário informa ao Engenho Ativo o nome da regra ativa a ter seu *status* alterado e escolhe se ele quer ativa-la ou desativa-la;

- Etapa de Mudança de *Status*: o Gerenciador de Regras monta o comando SQL de pesquisa e alteração da regra ativa informada e submete ao Oracle7 para que este efetue a mudança:

```
UPDATE Catalogo_Regras SET cr_status = 'A'
    WHERE cr_nome_regra = 'REGRA1';
```

No exemplo acima, a regra ativa REGRA1 criada nas seções anteriores, está sendo ativada.

## 5.3 Detalhes de Implementação do Monitor no Oracle7

Nesta seção, são descritos alguns detalhes da implementação do Monitor e de como é feito o processamento das Regras Ativas quando ocorre um evento. Antes de iniciar com os detalhes de implementação, vale relembrar a estrutura da Tabela Ativa apresentada na seção 5.1:

- procedimento-evento: nome do procedimento-evento;
- *status*-registro: indica se o registro já foi processado ou não pelo Monitor. Assume os valores: 'A' para "A Processar" ou 'P' para "Processado".

Para um melhor entendimento, esta atividade foi dividida em 7 etapas explicadas a seguir:

- Etapa de Inicialização: nesta etapa o usuário (ABD) executa o Monitor deixando-o em modo residente. A interface que deve aparecer para o usuário é idêntica à apresentada no capitulo anterior:


**Figura 5.5 – Interface do Monitor do Engenho Ativo versão para Oracle7**

- Etapa de Verificação de Eventos: iniciada automaticamente após a execução do Monitor. Periodicamente, o Monitor consulta a Tabela Ativa para verificar se existe algum registro que ainda não foi processado. O Monitor submete o seguinte comando SQL ao Oracle7 para fazer essa verificação (aqui também serão usados os nomes dos campos que foram utilizados no script de criação da Tabela Ativa apresentado no Anexo 3):

```
SELECT * FROM Tabela_Ativa WHERE ta_status_registro = 'A';
```

  Se a consulta acima não retornar nenhuma tupla então é por que não ocorreu nenhum evento e o Monitor deve voltar ao seu estado de espera. Se retornar pelo menos uma tupla, o Monitor inicia a próxima etapa;

- Etapa de Sinalização dos Eventos: o Monitor verifica se na expressão de eventos de cada regra ativa catalogada no Catálogo de Regras, existe o evento ocorrido. Ou seja, o Monitor lê o Catálogo de Regras e para cada regra ativa catalogada verifica se dentro da expressão de eventos (campo `cr_expr_eventos`) existe a cadeia presente no campo `ta_procedimento_evento` da Tabela Ativa. Se existir o evento simples é sinalizado (relembrar etapa de sinalização apresentada na seção 4.3.3);

- Etapa de Busca das Expressões Sinalizadas: após a Etapa de Sinalização dos Eventos, o Monitor varre o Catálogo de Regras em busca das expressões de eventos sinalizadas, ou seja, as expressões são avaliadas com base na gramática apresentada na seção 3.2.1. Cada evento simples sinalizado representa o valor TRUE, e os não sinalizados o valor FALSE. Daí, é só aplicar a gramática e verificar se o resultado da análise da expressão é verdadeiro ou falso (sinalizada ou não sinalizada). Para cada expressão

sinalizada, o Monitor verifica se a Regra Ativa correspondente está ativada, ou seja, com o campo `cr_status` assumindo o valor 'A' de ativa, e então insere a Regra Ativa na Tabela de Regras Candidatas;

- Etapa de Resolução de Conflitos: o Monitor organiza a Tabela de Regras Candidatas por ordem decrescente de prioridade;

- Etapa de Verificação das Condições: para cada Regra Ativa presente na Tabela de Regras Candidatas ordenada, o Monitor executa a condição associada, ou seja, executa o procedimento identificado pelo valor do campo `cr_condicao` que no caso do Oracle7 é uma chamada a uma *stored procedure* do tipo *function*, e verifica se o valor de retorno é TRUE ou FALSE. Se o valor retornado for TRUE a próxima etapa é iniciada, se não, a regra ativa é eliminada da Tabela de Regras Candidatas;

- Etapa de Execução das Ações: esta etapa consiste em descobrir se a ação associada é um procedimento armazenado, fazendo uma pesquisa no Dicionário de Dados, ou se é um aplicativo. Se for a chamada de um arquivo não executável, tipo .doc por exemplo, o Monitor abre o programa que reconhece e manipula com àquele tipo de arquivo. No caso do exemplo, o Word é chamado e o arquivo é aberto (ver seção 3.2.1 no que diz respeito as Ações).

Com o final da última etapa, o Monitor coloca o registro equivalente ao evento tratado como processado na Tabela Ativa, alterando para isso o campo `ta_status_registro` para 'P' e verifica a ocorrência de mais algum evento, ou seja, de mais algum registro com `ta_status_registro` = 'A'. Se existir, processa o evento seguindo as etapas descritas até que não haja mais nenhum registro a ser processado. Se não existir nenhum registro a ser processado, o Monitor entra em estado de espera durante o intervalo de tempo definido.

## 5.4 Conclusões

Os testes efetuados com a primeira versão do Engenho Ativo foram extremamente satisfatórios e, mesmo com as restrições adotadas, dotaram o Oracle 7.3 for Workgroups de características ativas bastante interessantes. Destacando: a possibilidade de definir eventos compostos inter-tabelas; e a capacidade de definir a ação de uma regra ativa como uma rotina externa, usando a linguagem preferida do usuário, em vez da linguagem proprietária do Oracle7, o PL/SQL.

Para utilizar o Engenho Ativo no topo de outro SGBD, o Informix por exemplo, não seriam necessárias muitas alterações no código do Engenho Ativo pois este funciona como uma camada que utiliza o SGBD como uma aplicação qualquer. As mudanças seriam decorrentes, principalmente, de possíveis diferenças na linguagem SQL implementada pelo Informix, já que o Engenho Ativo manipula com tabelas do BD, o Catálogo de Regras e a Tabela Ativa, por exemplo.

Concluindo, apesar de algumas restrições, a primeira versão do Engenho Ativo se mostrou eficiente e com um bom grau de portabilidade.

No próximo capítulo é apresentada uma aplicação desenvolvida utilizando o Engenho Ativo versão Oracle7.

# Aplicações do Engenho Ativo

Neste capítulo, é apresentado um exemplo de aplicação do Engenho Ativo na solução de um problema meteorológico de previsão do tempo. A aplicação consiste da automação da emissão de boletins meteorológicos para órgãos governamentais.

## 6.1 Aplicação de Meteorologia

No Laboratório de Meteorologia, Recursos Hídricos e Sensoriamento Remoto da Paraíba - LMRS-PB, sediado em Campina Grande, é feita, entre outras atividades, a previsão do tempo para todo o Estado da Paraíba. Nesta atividade, são gerados boletins meteorológicos que são enviados para os órgãos do governo estadual, com o fim de apoiar as decisões relacionadas à agricultura desenvolvida no estado.

O LMRS-PB mantém 210 estações meteorológicas, que fazem leitura diária do nível de precipitação de cada município. Estas estações enviam os dados diariamente, por meio de telefone, para o LMRS-PB, onde vão alimentar o Banco de Dados Hidro-Meteorológico. Esses dados servem de entrada para os boletins meteorológicos.

A emissão dos boletins obedece a um prazo limite. Às 10 horas da manhã, os boletins devem ser enviados aos órgãos concernentes. A aplicação deve verificar esta hora limite automaticamente, gerar e emitir automaticamente os boletins, e ainda avisar a um funcionário responsável quais estações ainda não enviaram os seus dados, para que providências cabíveis possam ser tomadas pelo responsável. No futuro próximo, serão instalados medidores automáticos, que enviarão os dados coletados diretamente para o banco de dados, via rede.

A aplicação é definida em termos das seguintes restrições: quando chegar a hora limite, a base de dados deve ser inspecionada, para verificar se todos os dados já foram enviados. Se faltar alguma estação, o sistema gera um relatório avisando quais estações faltam enviar dados, a fim de que o problema seja solucionado. Quando todos os dados tiverem sido enviados das estações, o sistema emite automaticamente os boletins, que são enviados por fax, também automaticamente, para os respectivos órgãos do governo.

### 6.1.1 Definição das Regras Ativas

A primeira regra ativa, R1, necessária ao sistema deve ser cadastrada na janela de criação de regras da seguinte maneira:

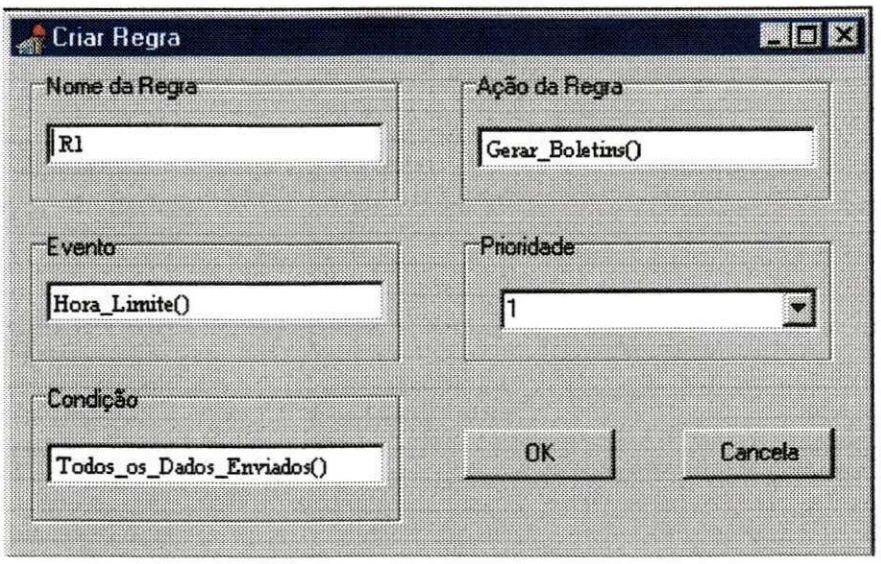



**Figura 6.1 – Criação da regra ativa R1**

Como se pode ver, a regra `R1` tem como evento o disparo do procedimento do usuário Hora_Limite(). O procedimento `Hora_Limite()` é chamado por uma rotina do usuário que fica testando se a hora do sistema operacional já chegou a 10:00 a.m.. Se sim, esta rotina chama o procedimento `Hora_Limite()`, caracterizando assim o evento da regra `R1`. No momento de criação de `R1`, o procedimento `Hora_Limite()` deve ter sido previamente preparado para caracterizar o evento, ou seja, o usuário, dono do procedimento, deve ter colocado no final do procedimento `Hora_Limite()`, o comando SQL:

```
INSERT INTO Tabela_Ativa
            (ta_procedimento_evento, ta_status_registro)
      VALUES ('Hora_Limite()', 'A');
```

O comando acima é o responsável por indiretamente avisar o Monitor que o evento `Hora_Limite()` ocorreu. Então, quando `Hora_Limite()` for executado, um novo registro será inserido na Tabela Ativa. Quando o Monitor verificar a existência desse novo registro na Tabela Ativa, ele inicia a procura, no Catálogo de Regras, das regras ativas associadas ao evento `Hora_Limite()` e acha `R1`. A etapa seguinte é a verificação da condição, executando o procedimento `Dados_Enviados()`, que consulta as tabelas do BD para ver se todos os dados já foram enviados. Se sim, o Monitor chama o procedimento ação `Gerar_Boletins()`.

O procedimento `Gerar_Boletins()`, por sua vez, dispara uma outra regra, R3, cuja ação é enviar, via fax, os boletins gerados. Essa regra deve ser criada da seguinte forma:

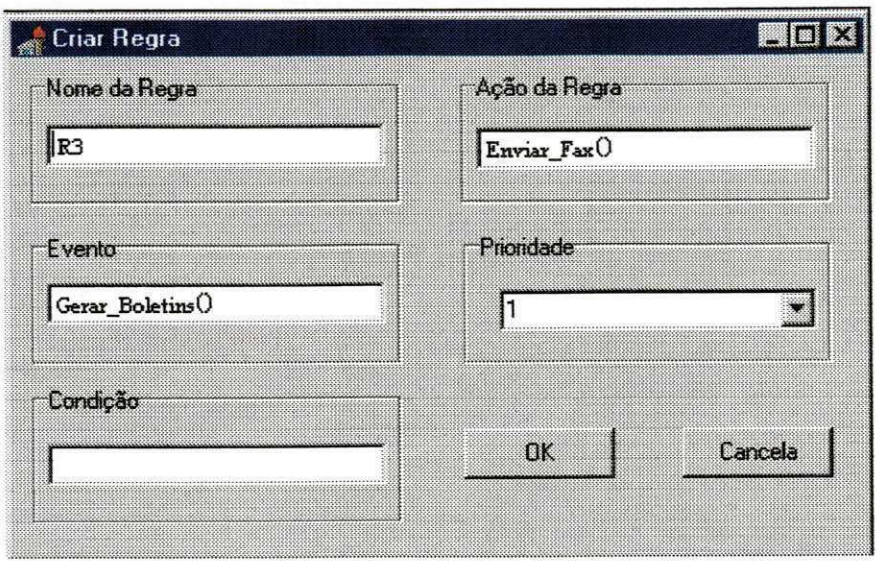



**Figura 6.2 – Criação da regra ativa R3**

Note que R3, além de ilustrar o aninhamento de regras, ou seja, a ação da regra R1 como evento da regra R3, serve também para mostrar a possibilidade de se definir regras ativas com condição verdadeira implicitamente.

Para que se possa avisar a quem de direito que todos os dados ainda não foram enviados pelas estações, é preciso criar uma outra regra ativa, a regra R2:

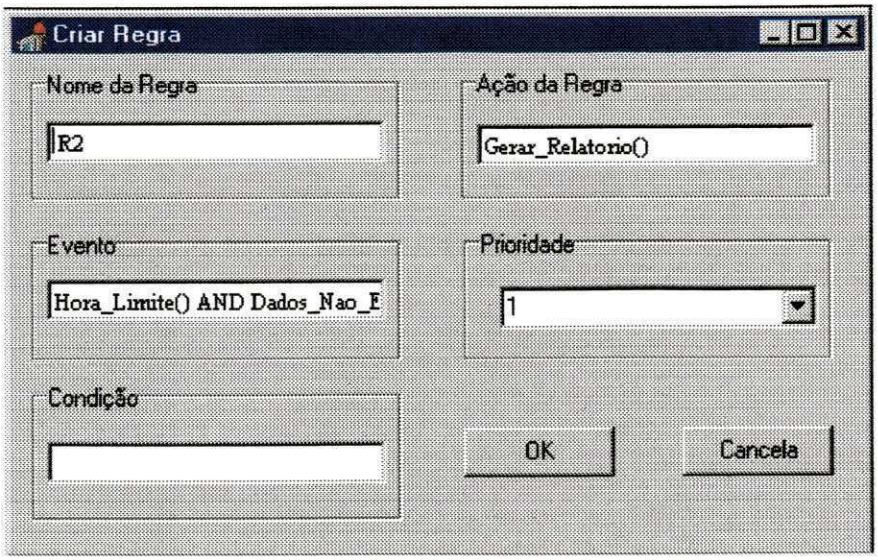



**Figura 6.3 – Criação da regra ativa R2**

A expressão de eventos definida pela regra R2 é uma expressão composta, na Figura 6.3 não ficou totalmente legível mas, ela foi definida como se segue:

```
Hora_Limite() AND Dados_Não_Enviados()
```

O procedimento `Hora_Limite()` é o mesmo que foi apresentado anteriormente. O procedimento `Dados_Não_Enviados()` é disparado por uma aplicação do usuário quando passa das 10:00 a.m. e todos os dados ainda não foram enviados. Quando o procedimento `Hora_Limite()` é chamado, ele avisa ao Monitor que ocorreu um evento, inserindo um registro na Tabela Ativa. Quando o Monitor consultar a Tabela Ativa ele irá disparar a regra `R1` mas não irá disparar a regra `R2` pois, sua expressão de eventos, depois de avaliada, não retornou um valor verdade. Quando o procedimento `Dados_Não_Enviados()` for chamado, ele também deve avisar ao Monitor que ocorreu um evento, inserindo um registro na Tabela Ativa. No momento em que o monitor verificar que existe um registro novo na Tabela Ativa e pesquisar as regras ativas associadas àquele evento, ele deverá achar `R2` e como o outro evento simples envolvido já aconteceu, no caso, a chamada ao procedimento `Hora_Limite()`, ele deve disparar a ação associada a `R2` já que a condição definida no momento de criação desta regra é sempre verdadeira.

Pode-se concluir que esta aplicação de meteorologia permite ressaltar as potencialidades do Engenho Ativo para dar apoio a atividades do mundo real, não convenientemente tratadas pelos SGBDs comerciais existentes.

O próximo capítulo apresenta as conclusões do presente trabalho e as propostas de trabalhos futuros.

# Conclusões e Propostas de Trabalhos Futuros

Este capítulo pretende sumariar as conclusões tiradas com a especificação do Engenho Ativo, comentar sua a implementação no topo do Oracle 7, e indicar alguns caminhos para possíveis aprimoramentos do Engenho Ativo.

## 7.1 Conclusões

A proposta de se oferecer um engenho capaz de dotar os SGBDs relacionais comerciais de novas funcionalidades ativas foi conseguida com um bom nível de portabilidade. Apesar de algumas restrições impostas pela arquitetura escolhida, pode-se dizer que o Engenho Ativo projetado atingiu, ao menos parcialmente, os objetivos que lhe foram atribuídos.

O Engenho Ativo suporta funcionalidades ativas muito importantes como a capacidade de definir eventos como chamadas a procedimentos do usuário, eventos envolvendo múltiplas tabelas e eventos compostos, funcionalidades estas que não estão presentes em praticamente nenhum SGBD presente no mercado.

A portabilidade foi conseguida pelo tratamento de eventos como procedimentos do usuário.

Com a abordagem da arquitetura em camadas, o Engenho Ativo foi implementado no topo do Oracle7, estendendo suas funcionalidades ativas, sem que fosse necessária nenhuma alteração no código fonte do Oracle7 e com relativamente pouco esforço de programação.

Entretanto, é sabido que o Engenho Ativo ainda apresenta muitas limitações. Tais limitações são tratadas na seção seguinte.

## 7.2 Limitações

Uma limitação importante surgida a partir da escolha da arquitetura em camadas para a implementação do Engenho Ativo diz respeito a perda da capacidade de se definir níveis de granularidade, linha e conjunto, para as regras ativas.

Nesta versão também não foi dada uma atenção maior para o tratamento e recuperação de erros. A única providencia tomada foi a definição das mensagens de erro retornadas pelos comandos de manipulação de regras ativas.

Mesmo atuando sobre SGBDs que oferecem tratamento de valores de transição, a especificação do Engenho Ativo não dá suporte à manipulação de tais valores. Esta, talvez, seja a limitação mais importante apresentada pelo Engenho Ativo. Porém, pode ser resolvida através da definição de tabelas temporárias que guardem os valores antigos e novos, como é feito no Sybase.

Pode-se ver claramente que a maioria das limitações pode ser facilmente resolvida, e que uma nova versão do Engenho Ativo poderia eliminar tais limitações. Quanto às limitações impostas pela arquitetura em camadas, com certeza a complexidade envolvida para solucionar tais limitações seria muito maior.

## 7.3 Propostas de Trabalhos Futuros

Além das limitações citadas anteriormente, algumas das quais podem ser facilmente resolvidas, outras funcionalidades podem ser adicionadas ao Engenho Ativo. Enumerando-as:

- Definição de eventos temporais. Eventos que promovem o disparo de regra(s) ativa(s) associada(s) em momentos pré-determinados. Dentre estes eventos pode-se citar: eventos de tempo absoluto, por exemplo, um evento será caracterizado quando chegar às 10:00 do dia 25 de dezembro; eventos repetitivos, por exemplo, um evento é caracterizado todos os dias às 18:30; e eventos repetitivos, por exemplo, de 5 em 5 minutos é caracterizado um evento;

- Definição de eventos compostos por sequência ou composição temporal. O primeiro tipo é caracterizado quando um evento é caracterizado quando dois ou mais eventos ocorrem em uma ordem particular. Por exemplo, E1 é definido pela expressão: E2 AND E3 nesta ordem, ou seja, E1 só ocorre se E2 e E3 ocorrem e E3 ocorre depois de E2. Já a composição temporal nada mais é do que um evento composto onde um dos eventos envolvidos é um evento temporal;

No caso de condições e ações, o Engenho Ativo oferece uma gama bastante razoável de construções.

# Bibliografia


[AMC93] Anwar, E.; Maugis, L.; Chakravarthy, S.: "*A new perspective on rule support for object-oriented databases*". Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 99-108.

[ASK92] ASK Computer Systems. "*INGRES/SQL Reference Manual, Version 6.4*". 1992.

[BP91] Bauzer, C., Pfeffer, P.: "Transformação de um Banco de Dados Orientado a Objetos para um Banco de Dados Ativo", 6° Simpósio Brasileiro de Banco de Dados, Manaus, Amazonas, 1991.

[CBB+89] Chakravarthy, S.; Blaustein, B.; Buchmann, A. P.; Carey, M.; Dayal, U.; Goldhirsch, D.; Hsu, M.; Jauhari, R.; Ladin, R.; Livny, M.; McCarthy, D.; McKee, R.; Rosenthal, A.: "*HiPAC: A research project in active,*

*time-constrained database management*". Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, Massachusetts, July 1989.

[CO92] Cannan, S. J.; Otten, G. A. M.: "*SQL-The Standard Handbook*". McGraw-Hill, London, 1992.

[CW96] Ceri, S.; Widom, J.: "*Active Database Systems: Triggers and Rules for Advanced Database Processing*", Morgan Kaufmann Publishers, San Francisco, California, 1996.

[DBM88] Dayal, U.; Buchmann, A. P.; McCarthy, D. R.: "*Rules Are Objects Too: A Knowledge Model For An Active, Object Oriented Database System*". Advances in Object-Oriented Database Systems, 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, FRG, September 27-30, 1988, Proceedings.

[D89] Date, C. J.: "*A Guide to the SQL Standard*". Second Edition, Addison-Wesley, Reading, MA, 1989.

[D81] Date, C. J.: "*Referential Integrity*". In Proceedings of the Seventh International Conference on Very Large Data Bases. Cannes, France, September 1981.

[GKBF98] Gatziu, Stella.; Koschel, A.; Bültzingsloewen, G. von; Fritschi, H.: "*Unbundling Active Functionality*". SIGMOD Record 27(1): 35-40 (1998).

[I94] Informix guide to SQL, Tutorial, Version 6, March 1994. Number 000-7598.

[JS94] Jagadish, H. V.; Shmueli, O.: "*Composite events in a distributed object-oriented database*". Distributed Object Management, Morgan Kaufmann Publishers, 1994, 248-268.

[KS93] Korth, H. F.; Silberschatz, A.: "Sistema de Bancos de Dados". Tradução Maurício Heihachiro Galvan Abe; revisão técnica Silvio Carmo Palmieri, 2ª Edição, MAKRON Books, São Paulo, 1993, 165-188.

[LW93] Lockemann, P. C.; Walter, H. D.: "*Activities in Object Bases*". Rules in Database Systems, N. W. Paton and M. H. Williams (Eds.), Springer-Verlag 1993.

[MS93] Melton, J.; Simon, A. R.: "*Understanding the New SQL: A Complete Guide*". Morgan Kaufmann Publishers, San Francisco, California, 1993, 125-147.

[SN97] Sampaio, M. C.; Nascimento, C. A.: "*Rule Support in a Object-Oriented Data Manager for Cooperative Design*". Anais/Simpósio Brasileiro de Bancos de Dados, Fortaleza, outubro 13-15, 1997, 287-301.

[S93] Silva, S. L. F.: "O Modelo Temporal de Objetos TOM", Relatório Técnico Científico do Departamento de Sistemas e Computação, Campina Grande, Paraíba, Outubro, 1995.

[SPAM91] Schreier, U.; Pirahesh, H.; Agrawal, R.; Moham, C.: "*Alert: An Architecture for Transforming a Passive DBMS into a Active DBMS*", Proceedings of the 17th International Conference on Very Large Data Bases, Barcelona, September, 1991.

[SYBASE] Transact-SQL user's guide for Sybase, release 10.0. Berkeley, California.

[ORACLE92] Bobrowski, S.: "*Oracle7™ Server Concepts Manual*", *Part Number* 6693-70-1292, Copyright Oracle Corporation, December, 1992.

[ZSF+98] Zaniolo, C.; Stefano, C.; Faloutsos, C.; Snodgrass, R. T.; Subrahmanian, V. S.; Zicari, R. "*Advanced Database Systems*", Morgan Kaufmann Publishers, San Francisco, California, 1998, 34-35.

# Anexos

# Anexo 1 - Algoritmos em Pseudo-código do Gerenciador de Regras

```
Gerenciador de Regras ()
INICIO
      SE atividade_desejada  =  Criar_regra ENTAO
            Abrir janela "Criar Regra"
      SENAO SE atividade_desejada  =  Excluir_regra ENTAO
            Abrir janela "Excluir Regra"
      SENAO SE atividade_desejada  =  Mudar status de regra ENTAO
            Chama tela "Status de Regra"
      SENAO SE atividade_desejada  =  Sair ENTAO
            Termina execução
FIM


FUNÇÃO Pesquisa_Regra(nome_regra)
INICIO
      Comando SQL pesquisando na tabela Catálogo de Regras se existe
      uma regra com o  mesmo nome.
      SE regra existir ENTAO
            retorna TRUE
      SENAO
            retorna FALSE
FIM
```

```
PROCEDIMENTO Criar_regra()
      nome_regra:              cadeia_de_caracteres;
      expressao_eventos:       cadeia_de_caracteres;
      condicao:                cadeia_de_caracteres;
      ação:                    cadeia_de_caracteres;
      data_criação:            data;
      prioridade:              inteiro;
      achou:                   boolean;

INICIO
      Ler_atributos_nova_regra(nome_regra, expressao_eventos,
      condicao, ação, prioridade, data_criacao);

      achou = Pesquisa_Regra(nome_regra);
      SE achou = verdade ENTAO
      INICIO
            Erro: Regra já existe
            RETORNAR;
      FIM
      SENAO
      INICIO
            expressao_eventos =
      Sinaliza_expr_eventos(expressao_eventos);
            Criar_Nova_Regra( nome_regra, expressao_eventos, condicao,
                              ação , prioridade,data_criacao);
      FIM
FIM


FUNÇÃO  Sinaliza_expr_eventos(expr_eventos) : expr_final;
      objeto:     cadeia_de_caracteres;
      operador:   cadeia_de_caracteres;
      token :     cadeia_de_caracteres;
      aux:        cadeia_de_caracteres;

INICIO
      I = 0;
      caracter = Le_caracter();
      ENQUANTO (expr_eventos <> FIM) FACA
      INICIO
            SE caracter = '(' ENTAO
                  expr_final = expr_final + caracter;
            SENAO SE caracter = 'A' OU caracter ='N' OU caracter = 'O'
            ENTAO
            INICIO
                  SE (caracter = 'A' OU caracter ='N') ENTAO
                        operador = caracter + próximos dois caracteres
                  SENAO
                        operador = caracter + próximo caracter
                  expr_final = expr_final + operador;
            FIM
            SENAO SE (caracter <> ')') ENTAO
            INICIO
                  ENQUANTO (caracter <> '(' ou branco) FACA
                  INICIO
                        Token = token + caracter;
                        expr_final = expr_final + caracter;
                        caracter = Le_caracter;
                  FIM
                  ENQUANTO(caracter <> ')'  ou  branco) faça
```

```
                INICIO
                      objeto = objeto + caracter;
                      caracter  =  Le_caracter;
                FIM
                expr_final = expr_final + objeto;
                expr_final = expr_final + ')';
                expr_final = expr_final + '-';
          SENAO SE (caracter = ')') ENTAO
                expr_final = expr_final + caracter;
          caracter = Le_caracter;
    FIM
    Retorna (expr_final);
FIM


PROCEDIMENTO Excluir_Regra()
    nome_regra:       cadeia_de_caracteres;
    expr_eventos:     cadeia_de_caracteres;
    objeto:           cadeia_de_caracteres;
    operação:         cadeia_de_caracteres;
    excluir_trigger:  boolean;

INICIO
    nome_regra = Le_nome_regra();
    achou = Pesquisa_Regra (nome_regra);
    SE achou ENTAO
    INICIO
        Excluir a regra com nome = nome_regra do Catalogo de
            Regras;
    FIM
    SENAO
    INICIO
        Erro: Regra desejada não existe;
        RETORNAR;
    FIM
FIM


PROCEDIMENTO Status_Regra()
    nome_regra :      cadeia_de_caracteres;
    ativa :           boolean;

INICIO
    achou = Pesquisa_regra(nome_regra);
    SE achou  = FALSE ENTAO
    INICIO
        Erro: Regra não existe;
        RETORNAR;
    FIM
    SENAO SE ativa = TRUE ENTAO
    INICIO
        Mudar status de nome_regra para ATIVA;
        RETORNAR;
    FIM
    SENAO
    INICIO
        Mudar status de nome_regra para DESATIVA;
        RETORNAR;
    FIM
FIM
```

# Anexo 2 - Algoritmos em Pseudo-código do Monitor

```
FUNCAO get_token : cadeia_de_caracteres
      v_cadeia_aux :    cadeia_de_caracteres
      v_caracter_aux :  caracter
      v_len :           inteiro
INICIO
      v_cadeia_aux = ''
      SE v_expr_eventos = '' ENTAO
            RETORNAR ' '
      FIM SE
      SE ( v_expr_eventos[1] <> 'A' ) AND ( v_expr_eventos[1] <> 'N' )
                  AND ( v_expr_eventos[1] <> 'O' )
                  AND ( v_expr_eventos[1] <> '(' )
                  AND ( v_expr_eventos[1] <> ')' )
                  AND ( v_expr_eventos[1] <> '-' )
                  AND ( v_expr_eventos[1] <> '+' )

      ENTAO
      INICIO
            v_len = 1

            ENQUANTO ( v_expr_eventos[v_len] <> ')' ) FACA
                  v_len = v_len + 1

            SE ( v_expr_eventos[v_len + 2] = #0 ) ENTAO
                        v_cadeia_aux = ''
                  SENAO
```

```
                    Copiar para v_cadeia_aux, v_expr_eventos a
                          partir da posição( v_len + 2 ) até o
                          final.

        SE v_expr_eventos[v_len + 1] = '+' ENTAO
        INICIO
              v_expr_eventos = v_cadeia_aux
              RETORNAR 'TRUE'
        FIM
        SENAO
        INICIO
              v_expr_eventos = v_cadeia_aux
              RETORNAR 'FALSE'
        FIM
FIM
SENAO SE ( v_expr_eventos[1] = 'A' ) AND
              ( v_expr_eventos[2] = 'N' ) AND
              ( v_expr_eventos[3] = 'D' ) ENTAO
INICIO
        v_len = 3
        SE ( v_expr_eventos[v_len + 1] = #0 ) ENTAO
              v_cadeia_aux = ''
        SENAO
              Copiar para v_cadeia_aux, v_expr_eventos a partir de
                    (v_len + 1) até o final.
        v_expr_eventos = v_cadeia_aux
        RETORNAR 'AND'
FIM
SENAO SE ( v_expr_eventos[1] = 'O' ) AND
              ( v_expr_eventos[2] = 'R' ) ENTAO
INICIO
        v_len := 2
                    SE ( v_expr_eventos[v_len + 1] = #0 ) ENTAO
              v_cadeia_aux = ''
                    SENAO
                          Copiar para v_cadeia_aux, v_expr_eventos
                                a partir de ( v_len + 1 ) até o
                                final
        v_expr_eventos := v_cadeia_aux;
        RETORNAR 'OR'
FIM
SENAO SE ( v_expr_eventos[1] = 'N' ) AND
              ( v_expr_eventos[2] = 'O' ) AND
              ( v_expr_eventos[3] = 'T' ) ENTAO
INICIO
        v_len = 3
        SE ( v_expr_eventos[v_len + 1] = #0 ) ENTAO
              v_cadeia_aux = ''
        SENAO
              Copiar para v_cadeia_aux, v_expr_eventos a partir de
                    ( v_len + 1 ) até o final
        v_expr_eventos = v_cadeia_aux
        RETORNAR 'NOT'
FIM
SENAO
INICIO
        v_caracter_aux = v_expr_eventos[1]
        SE ( v_caracter_aux = '(' ) OU ( v_caracter_aux = ')' )
ENTAO
        INICIO
              SE ( v_expr_eventos[2] = #0 ) ENTAO
```

```
                        v_cadeia_aux = ''
                  SENAO
                        Copiar para v_cadeia_aux, v_expr_eventos a
                              partir da posição 2 até o final
                        v_expr_eventos = v_cadeia_aux
                        RETORNAR v_caracter_aux;
            FIM
            SENAO
                        RETORNAR ' '
      FIM
FIM.

{-----------}

FUNCAO fator_logico : BOOLEAN
      v_result : BOOLEAN
      v_token : cadeia_de_caracteres

INICIO
      v_token = get_token()
      SE ( v_token = 'TRUE' ) ENTAO
            RETORNAR TRUE
      SENAO SE ( v_token = 'FALSE' ) ENTAO
            RETORNAR FALSE
      SENAO
      INICIO
            SE ( v_token = '(' ) ENTAO
            INICIO
                  v_result = expr_logica()
                  v_token = get_token()
            FIM
            SENAO SE ( v_token = 'NOT' ) ENTAO
                  RETORNAR NOT( fator_logico() )
      FIM
FIM.

{-----------}

FUNCAO termo_logico : BOOLEAN
      v_result_esq : BOOLEAN
      v_result_dir : BOOLEAN
      v_token : cadeia_de_caracteres
INICIO
      v_result_esq = fator_logico()
      v_token = get_token()
      SE ( v_token <> ' ' ) ENTAO
            SE ( v_token <> 'AND' ) ENTAO
                  v_expr_eventos := v_token + v_expr_eventos
            SENAO
                  ENQUANTO (v_token = 'AND' ) FACA
                  INICIO
                        v_result_dir = fator_logico()
                        RETORNAR (v_result_esq AND v_result_dir)
                  FIM
                  RETORNAR v_result_esq
FIM.

{-----------}

FUNCAO expr_logica : BOOLEAN
      v_result_esq : BOOLEAN
```

```
      v_result_dir : BOOLEAN
      v_token : STRING

INICIO
      v_result_esq = termo_logico()
      v_token = get_token()
      SE ( v_token <> ' ' ) ENTAO
      ENQUANTO ( v_token = 'OR' ) FACA
      INICIO
            v_result_dir = termo_logico()
            RETORNAR (v_result_esq OR v_result_dir)
      FIM
      RETORNAR v_result_esq
FIM.

{-----------}

FUNCAO arvore_sinalizada( expr_eventos : cadeia_de_caracteres) :
BOOLEAN
INICIO
      v_expr_eventos = expr_eventos
      RETORNAR expr_logica()
FIM.

{-----------}

      PROCEDIMENTO selecionar_regras( procedimento_evento :
cadeia_de_caracteres);

      v_expr_pesq :           cadeia_de_caracteres
      v_expr_aux :            cadeia_de_caracteres
      v_expr_pos :            inteiro
      v_executar_acao :       boolean

INICIO
      v_expr_pesq = ''
      v_expr_aux = ''
      v_expr_pesq = procedimento_evento + '-';
      Ir para o inicio do Catalogo_de_Regras
      REPETIR
            v_expr_aux = Catalogo_de_Regras.cr_expr_eventos
            {* Sinalizar evento dentro da expressão de eventos *}
            v_expr_pos := Posição de v_expr_pesq dentro de v_expr_aux
            SE ( v_expr_pos > 0 ) ENTAO
            INICIO
                  SE Catalogo_de_Regras.cr_status = 'A' ENTAO
                  INICIO
                        Trocar o '-' associado a v_expr_pesq dentro de
                              v_expr_aux por '+'
                        Catalogo_de_Regras.cr_expr_eventos = v_expr_aux

                        {* Obter regras candidatas *}
                        SE (arvore_sinalizada(v_expr_aux)) ENTAO
                        INICIO
                              Trocar todos os '+' de
                                    Catalogo_de_Regras.cr_expr_eventos
                                    por '-'
                              Inserir a Regra Ativa na Tabela de Regras
                                    Candidatas
                        FIM
                  FIM
```

```
            FIM
            Ir para o próximo registro de Catalogo de Regras
      ATE o fim de Catalogo de Regras

      {* Incorporação do Resolvedor de Conflitos *}

      Organizar em ordem decrescente a Tabela de Regras Candidatas por
            prioridade e data de criacao

      ENQUANTO não for o fim da Tabela de Regras Candidatas FACA
      INICIO
            SE Regra Ativa não tiver condicao OU
                  condicao da Regra Ativa voltar TRUE ENTAO
                  Executar acao da Regra Ativa
            Ir para o próximo registro de Tabela de Regras Candidatas
      FIM

      Limpar a Tabela de Regras Candidatas
FIM.

{-----------}

PROCEDIMENTO monitor
      v_achou :   BOOLEAN

INICIO
      ENQUANTO TRUE FAÇA
            SE botao finalizar pressionado ENTAO
                  Saia do programa.
            SE está na hora de ver a Tabela Ativa ENTAO
            INICIO
                  SE existem registros na Tabela Ativa com
                        ta_status_registro = 'A' ENTAO
                  INICIO
                        selecionar_regras(ta_procedimento_evento)
            FIM
      FIM ENQUANTO
FIM
```

# Anexo 3 - Estrutura das Tabelas

As tabelas Catálogo de Regras e Regras Candidatas a Execução foram implementadas no protótipo do Engenho Ativo para Oracle7. Abaixo segue os *scripts* utilizados para criação destas tabelas. Note que a tabela `Regras_Candidatas` possue dois campos que não foram mencionados na especificação e que foram inseridos quando da implementação do protótipo a fim de melhorar a performance. Os campos `rc_condicao` e `rc_acao` significam, respectivamente, o nome da função que atua como condição da regra e o nome do procedimento que atua como ação da regra.

```
CREATE TABLE Catalogo_Regras (
  cr_nome_regra   VARCHAR2(50)      CONSTRAINT ccr_nome_regra NOT NULL,
  cr_condicao     VARCHAR2(255),
  cr_acao         VARCHAR2(255)     CONSTRAINT ccr_acao NOT NULL,
  cr_status       VARCHAR2(1)       CONSTRAINT ccr_status NOT NULL,
  cr_prioridade   NUMBER(2, 0)      CONSTRAINT ccr_prioridade NOT NULL,
  cr_expr_eventos VARCHAR2(255)  CONSTRAINT ccr_expr_eventos NOT NULL,
          CONSTRAINT pk_cr_nome_regra PRIMARY KEY (cr_nome_regra)
)
TABLESPACE user_data;
```

```
CREATE TABLE regras_candidatas
(rc_nome_regra    VARCHAR2(50)      CONSTRAINT crc_nome_regra NOT NULL,
 rc_condicao      VARCHAR2(255),
 rc_acao          VARCHAR2(255)     CONSTRAINT crc_acao NOT NULL,
 rc_prioridade    NUMBER(2)   CONSTRAINT crc_prioridade NOT NULL
)
TABLESPACE user_data;

CREATE TABLE Tabela_Ativa
(ta_procedimento_evento       VARCHAR2(50)
            CONSTRAINT cta_procedimento_evento NOT NULL,
 ta_status_registro     CHAR(1)
            CONSTRAINT cta_status_registro NOT NULL
            CONSTRAINT pk_ta_procedimento_evento PRIMARY KEY
                                        (ta_procedimento_evento)
)
TABLESPACE user_data;
```