UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Guilherme Huth

Um modelo para o gerenciamento de bancos de dados SQL através de *Stored Procedures*

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Elizabeth Sueli Specialski

Florianópolis, Dezembro de 2002

Um modelo para o gerenciamento de bancos de dados SQL através de *Stored Procedures*

Guilherme Huth

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Fernando Ostuni Gauthier, Dr.
Coordenador

Banca Examinadora

_____

Profª. Elizabeth Sueli Specialski, Drª.
Orientadora

_____

Prof. Vitório Bruno Mazzola, Dr.

_____

Prof. Alexandre Moraes Ramos, Dr.

"Só fazemos melhor aquilo que repetidamente insistimos em melhorar.
A busca da excelência não deve ser um objetivo. E, sim, um hábito."

(Aristóteles)

Agradecimentos

Inicialmente, aos professores de um modo em geral que dedicam seus esforços na disseminação do conhecimento, na formação da cultura de um povo e, principalmente, na construção do caráter do ser humano.

À minha orientadora, professora Elizabeth Sueli Specialski, que exerceu com firmeza, objetividade e franqueza o seu papel e foi de extrema importância para a realização deste trabalho.

Ao meu líder e amigo, Gilson Norberto Horstmann, pelo interesse demonstrado e pelas palavras de apoio e incentivo durante a elaboração desta tese.

Em especial à minha esposa Marli, companheira amorosa e sensível, pela sua compreensão e resignação durante minhas ausências necessárias à realização deste trabalho.

Ao meu filho Gustavo André que constantemente teve que reclamar a presença do pai.

À minha filha Amanda Carolina que, após o seu nascimento, teve que somar os seus protestos aos do irmão.

E à DEUS, por ter reunido todas estas pessoas e tê-las feito parte de minha vida.

Sumário

Resumo

Este trabalho apresenta uma proposta de modelo para gerenciamento de bancos de dados relacionais que utilizam a linguagem SQL como padrão. Foram estudadas as características gerais dos bancos de dados SQL para que a definição do modelo atendesse o formato genérico, ou seja, fosse aplicável para qualquer SGBD SQL. A flexibilização do módulo de gerenciamento foi o principal objetivo na definição do modelo proposto, tendo como principal foco a definição de regras de gerenciamento customizáveis. A validação do modelo foi realizada através de uma implementação em um ambiente de bancos de dados (utilizando o SGBD Oracle). Os resultados desta implementação foram comparados com as funcionalidades de um produto de mercado (Unicenter TNG) como forma de avaliação da eficiência do modelo proposto.

Abstract

This work presents a proposal for a model management of relational databases that uses the SQL language as standard. The general characteristics of the SQL databases were studied to attend a model definition in a generic format, that is, applicable for any SQL DBMS. The main objective in the definition of the proposed model was the flexibility in the management model, having as main focus the definition of tailored management rules. The validation of the model was carried out with an implementation in a database environment (utilizing the Oracle DBMS). The results of this implementation were compared with the features of a management product (Unicenter TNG) as a form of efficiency evaluation for the proposed model.

1 Introdução

1.1 Apresentação

Os ambientes computacionais atuais vêm suportando uma gama considerável de aplicações. Processos administrativos e fabris estão cada vez mais dependentes de sistemas informatizados. Nesse panorama, a utilização de bancos de dados tem se tornado uma constante. Desta forma, o gerenciamento destas bases torna-se vital para a disponibilidade, segurança, integridade e desempenho destas aplicações. Para auxiliar nesta tarefa de gerenciamento são adotados produtos (*softwares*) disponíveis no mercado. Estes produtos, apesar de sua grande utilidade, nem sempre conseguem atender todas as expectativas (que são bastante variadas e particulares a cada ambiente).

Uma pesquisa no mercado traria como resultado uma quantidade considerável de produtos de gerenciamento para bancos de dados. Não se pretende esgotar todas as opções possíveis e muito menos realizar uma avaliação detalhada de cada alternativa. O objetivo é demonstrar a forma como estas alternativas são implementadas para confrontá-las com o modelo proposto neste trabalho. Com este intuito, serão apresentadas as principais desvantagens que podem estar presentes nos produtos de mercado.

Ao iniciarmos o questionamento destas desvantagens, um dos aspectos a ser considerado é a dependência do fornecedor. É preciso comparar a facilidade em adquirir um produto de mercado com o esforço que seria despendido no projeto, desenvolvimento e implementação de uma ferramenta "caseira". Contudo, também é preciso considerar o grau de dependência criado com este fornecedor e os eventuais atrasos no incremento/correção do produto adquirido.

Outra característica dos produtos de mercado é a necessidade de instalação e configuração em todos os clientes e gerente(s) (cada produto demanda aparatos e infra-estrutura própria). Atualizações/correções no produto sempre demandam a tradicional maratona para equalização das versões instaladas nos mais diversos equipamentos. Dentre as necessidades de equalização constante do produto destacam-se as atualizações/correções necessárias em Gerentes, Agentes, Repositório de Dados, DLL (*Dynamic Link Library*), MIB (*Management Information Base*), *Front-end,* Pacotes (*Packages*) específicos de gerenciamento e métodos proprietários de comunicação.

A customização, normalmente rígida, das regras de gerenciamento é outro ponto desfavorável. Existem situações em que a forma de medição padrão não atende as necessidades reais ou, no mínimo, não traz a eficiência desejada. Existem produtos que contornam esta desvantagem, mas existem muitos que exigem a aquisição de produtos complementares ou não apresentam esta alternativa em hipótese alguma.

Falhas na comunicação entre os módulos, agentes desatualizados ou inativos, console central inoperante, etc. Problemas da natureza dos exemplos citados tendem à comprometer o gerenciamento dos ambientes. Produtos pouco robustos ou que não tenham um mecanismo de auto-verificação podem deixar de coletar informações ou de emitir alertas necessários.

1.2 Motivação

Os produtos de gerenciamento, normalmente, apresentam as suas próprias regras pré-definidas. A primeira desvantagem defrontada é o número normalmente reduzido destas regras de gerenciamento. Se forem consideradas a gama de objetos e as situações presentes em um ambiente de banco de dados, observa-se que estas regras pré-definidas disponibilizadas se limitam a poucos objetos ou situações passíveis de gerenciamento. E, se apenas este número reduzido de regras não apresentasse uma grande dificuldade, ainda soma-se a isto a inflexibilidade das mesmas. É muito comum se encontrar regras rigidamente atadas ao produto oferecido, havendo pouca ou nenhuma possibilidade de customização. A adoção dos valores *default* não é adequada para todos os casos e, a alteração em cada objeto (individualmente), demanda bastante esforço. Objetos com padrões de comportamento diferenciado ao longo do tempo demandam a necessidade de uma revisão e acompanhamento constantes de seus valores referenciais.

Ainda focando a questão das regras, observa-se que existe a situação oposta que também traz dificuldades para sua implementação. Alguns produtos, na tentativa de suprir todas as eventuais necessidades de gerenciamento, apresentam uma gama enorme de regras pré-definidas. Centenas de regras são disponibilizadas com este intuito, mas, se por um lado muitas necessidades são atendidas com esta estratégia, em contrapartida a enorme quantidade dificulta a avaliação e identificação das regras efetivamente necessárias. Mesmo assim, não existe a garantia de que todas necessidades serão supridas em plena conformidade com cada situação. Cada ambiente traz as suas particularidades e nuanças que variam muito para cada caso. Ainda que exista a regra disponível para uma determinada necessidade, não se está isento de encontrar deficiências ou rigidez na sua construção que não atendam completamente às necessidades.

As deficiências apontadas acima desencadeiam, normalmente, a aquisição de outros produtos ou módulos complementares para o gerenciamento. Alguns fornecedores atentos à necessidade de uma maior flexibilização, disponibilizam ferramentas que permitem a construção destes módulos complementares ou o desenvolvimento de novos produtos. Se, por um lado, esta alternativa parece suprir todas as necessidades particulares de cada caso, em contrapartida, a solução implementada é totalmente dependente do fornecedor. Além de depender do fornecedor para a correção dos problemas comuns ao produto será criada a dependência com relação à liberação de novas *releases* (que acompanhem a evolução dos objetos gerenciados). A substituição do produto por outro similar, de outro fabricante,

invariavelmente invalida todo o trabalho de customização e desenvolvimento complementar realizado. Atualizações do produto, mesmo que oriundas do próprio fornecedor, também podem acarretar a invalidação ou incompatibilidade desta complementação desenvolvida.

As deficiências descritas acima, confrontadas com o propósito de obtenção de um modelo de gerenciamento genérico para bases de dados SQL foram fatores motivadores deste trabalho. A possibilidade da incorporação deste modelo ao próprio SGBD e a concepção de um modelo que permitisse a definição de regras de gerenciamento customizáveis também foram aspectos motivadores que contribuíram para o desenvolvimento deste trabalho.

1.3 Delimitação

No universo de banco de dados relacionais foram desenvolvidas linguagens destinadas à sua manipulação, sendo que a SQL (*Structured Query Language*) foi a de maior aceitação. *Oracle*, *Microsoft SQL Server*, *Sybase*, *Informix* e *MySQL* são exemplos de gerenciadores de bancos de dados que adotaram esta linguagem.

Apesar de não existir uma padronização de fato (SQL-92 é o padrão mais utilizado) as pequenas diferenças de sintaxe não comprometem a portabilidade de aplicações desenvolvidas em SQL. Desta forma, a linguagem SQL será utilizada para o desenvolvimento do modelo de gerenciamento a ser proposto (visando garantir a sua portabilidade).

Um banco de dados SQL fornece um conjunto de tabelas e visões contendo informações relacionadas ao próprio banco de dados, denominado dicionário de dados. Este dicionário contém informações sobre os objetos existentes, alocação de espaços, valores *default* para as colunas, regras de integridade referencial, usuários (nomes, privilégios e *roles* associadas), parâmetros de inicialização, arquivos do banco de dados, segmentos de *rollback*, sessões correntes, informações sobre desempenho do banco de dados, etc.

Outra característica presente nos SGBD´s SQL é a possibilidade de criação de *stored procedures*. O agrupamento de comandos SQL armazenados no dicionário de dados permitirá a incorporação do modelo de gerenciamento ao próprio SGBD, dotando-o, inclusive, de outra característica: a interoperabilidade.

1.4 Objetivos

O objetivo geral deste trabalho é desenvolver um modelo de gerenciamento para banco de dados SQL que permita a definição de regras customizáveis através de *stored procedures* (apresentando flexibilidade na composição de regras de gerenciamento).

Para o alcance deste objetivo, estabeleceu-se os seguintes objetivos específicos:

- Identificação e avaliação dos benefícios da utilização de *stored procedures*.

- Avaliação dos modelos de gerenciamento de redes (visão macro).

- Descrição de um modelo atual (produto de mercado).

- Mapeamento do modelo proposto (adequação ao modelo genérico de gerenciamento de redes).

- Definição dos componentes e características principais do modelo proposto.

- Detalhamento do modelo de dados.

- Implementação do modelo proposto em ambiente apropriado.

- Avaliação dos resultados obtidos através de comparativo com o modelo atual.

Com a implementação do modelo proposto pretende-se realizar sua validação, ou seja, garantir que as definições do modelo conceitual sejam aplicáveis ao ambiente de implementação selecionado. Através do comparativo com o modelo atual será avaliado o grau de eficiência do modelo, permitindo a emissão de críticas sobre o modelo proposto.

1.5 Organização do Trabalho

Este trabalho está organizado em 8 capítulos. O capítulo 2 apresenta as características básicas sobre bancos de dados SQL e as vantagens da utilização de SQL *Stored Procedures*. Descreve, sucintamente, os modelos de gerenciamento de redes e os objetivos da administração de bancos de dados. O detalhamento do modelo adotado por uma solução de mercado (Unicenter TNG) é realizado para permitir uma posterior comparação com o modelo proposto.

O capítulo 3 apresenta a alternativa de gerenciamento para bancos de dados SQL proposta neste trabalho. O mapeamento do modelo é realizado, mostrando a adequação realizada para o modelo genérico de gerenciamento de redes. A visão geral do modelo proposto é apresentada, são detalhadas suas características, o modelo de dados é explanado e suas principais vantagens são relacionadas.

A viabilidade de implementação do modelo de gerenciamento proposto é apresentada no capítulo 4. É realizada a descrição do ambiente de desenvolvimento e o detalhamento do processo de instalação e customização do módulo de gerenciamento.

Os resultados obtidos com o modelo implementado são apresentados no capítulo 5, efetuando-se um comparativo com a situação atual.

O capítulo 6 contém as avaliações do modelo proposto e descreve as conclusões sobre a implementação do módulo de gerenciamento. São destacados os pontos fortes e emitidas considerações sobre melhorias e sugestões para a continuidade deste trabalho.

Encerrando este trabalho, o capítulo 7 traz a bibliografia que forneceu a base para as definições e proposições desta tese de mestrado e o capítulo 8 (Anexos) apresenta exemplos de regras customizadas (*Stored Procedures*).

2 Gerenciamento de Bancos de Dados SQL

Este capítulo descreve o surgimento da linguagem SQL através de um breve histórico. Destaca a facilidade de aprendizado e também apresenta os benefícios da utilização de *stored procedures* nesta linguagem. Descreve as arquiteturas de gerenciamento de redes e também apresenta uma breve descrição da administração de banco de dados, esclarecendo as necessidades de gerenciamento. O modelo utilizado pelo produto Unicenter TNG será descrito em maiores detalhes, haja visto que será referenciado no decorrer deste trabalho, realizando-se um comparativo com o modelo proposto.

2.1 Structured Query Language - SQL

2.1.1 Introdução

O Dr. E. F. Codd, pesquisador da IBM, foi a primeira pessoa à publicar um artigo sobre bancos de dados relacionais (década de 70). Esta idéia desenvolveu-se através de um projeto da IBM, gerando a linguagem denominada inicialmente de SEQUEL (acrônimo para *Structured English Query Language*). Mais tarde, por razões legais, a denominação foi alterada para SQL simplesmente (embora muitos, atualmente, ainda pronunciem o nome original quando se referem à esta linguagem) [01].

No ano de 1974 a IBM tinha um protótipo do SGBD denominado SYSTEM/R. Este projeto foi concluído em 1979 tendo comprovado a viabilidade do modelo de dados relacional e contribuído significativamente para o desenvolvimento da linguagem SQL. Neste mesmo ano foi lançado no mercado o primeiro SGBD utilizando a linguagem SQL (este produto era o resultado do trabalho de uma equipe que avaliou minuciosamente o SYSTEM/R da IBM). Este produto foi chamado de Oracle. Desde então, outros SGBD´s foram sendo lançados no mercado (a grande maioria suportando a linguagem SQL que tornou-se padrão de fato para bancos de dados relacionais) [02].

2.1.2 Sintaxe

O modelo de dados SQL utiliza tabelas (compostas por colunas e linhas, representando os campos e os registros, respectivamente). Os relacionamentos entre as tabelas são efetuados por intermédio das colunas. A visualização das linhas é feita de forma intuitiva e totalmente diferente (comandos desta linguagem permitiram a manipulação de coleções de registros). A programação se tornou mais fácil e a quantidade de códigos a serem gerados diminuiu consideravelmente [03].

Conforme comentado anteriormente, a linguagem SQL tornou-se o padrão para a maioria dos bancos de dados relacionais disponibilizados no mercado. Sua grande aceitação

deve-se ao fato de ser uma linguagem de fácil aprendizado, ao mesmo tempo em que provê todas as necessidades de administração e manutenção dos bancos de dados. Sua estrutura assemelha-se ao Inglês, permitindo a intuição dos comandos visualizados (mesmo que a pessoa não seja especialista). Todos os comandos seguem a mesma estrutura básica, iniciando com um verbo que indica a operação a ser executada. Os comandos mais comuns são os destinados à manipulação de dados (*SELECT*, *INSERT*, *DELETE* e *UPDATE*). Após o verbo indicativo da ação desejada devem ser informados os campos das tabelas e as tabelas especificamente. Ainda podem ser utilizadas cláusulas para filtro (*WHERE*), ordenação (*ORDER BY*) e agrupamento (*GROUP BY*) entre outras alternativas que compõem a sintaxe do comando [04].

2.2 Stored Procedures

2.2.1 Descrição

Uma *stored procedure* é composta por um ou mais comandos SQL agrupados e armazenados no próprio SGBD. Normalmente cada *stored procedure* é criada para suportar a execução de determinada função ou tarefa repetitiva.

Podem existir diversos parâmetros de entrada ou mesmo nenhum. Os valores passados em tempo de execução são recebidos pela *stored procedure* e armazenados em variáveis internas (que devem ser definidas com o tipo e tamanho apropriados) [05].

Através destes parâmetros de entrada (informados no momento da execução) a *stored procedure* recebe valores que serão utilizados no processamento (seqüência de comandos SQL) que produzirá um determinado resultado. A aplicação que iniciou a execução da *stored procedure* poderá receber este resultado através de uma lista de registros, parâmetros de saída ou código de retorno [06].

Ao criar uma *stored procedure* são escritos dois códigos distintos. O primeiro é a codificação da própria *stored procedure* que ficará armazenada no SGBD. O segundo código refere-se ao aplicativo que será executado na estação do cliente ou mesmo em um servidor *middleware* (*Web Server*, por exemplo). A aplicação, neste caso, somente precisa fazer a chamada da *stored procedure* previamente definida [07]. A figura 2.1 descreve a diferença entre uma aplicação convencional e a utilização de *stored procedures*:

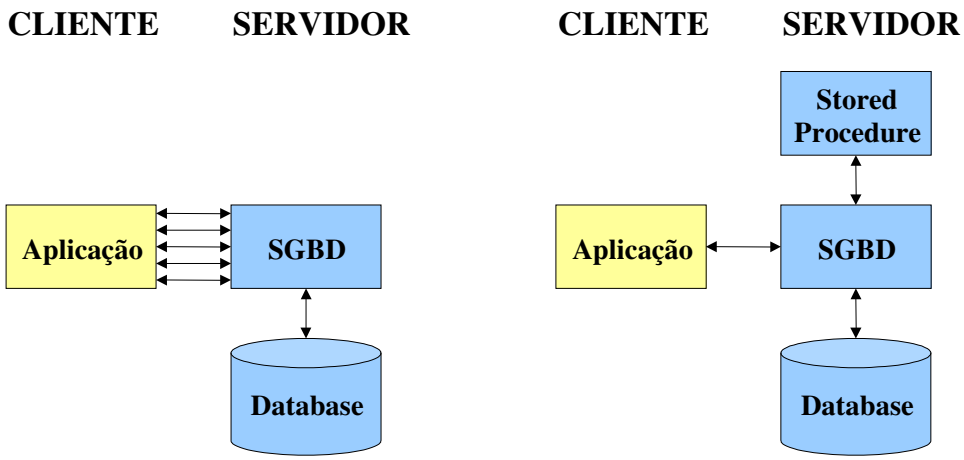

Figura 2.1 – Aplicação convencional X Utilização de *stored procedure*

2.2.2 Benefícios

Abaixo são apresentados alguns benefícios da utilização de *stored procedures*:

Redução do tráfego de rede entre estação cliente e servidor:

- Quando uma aplicação que está executando na estação cliente passa uma requisição para uma *stored procedure* residente no servidor de SGBD ocorre uma redução de tráfego na rede. A *stored procedure* fará todo o processamento intermediário diretamente no servidor e retornará à aplicação apenas os registros resultantes. Se o processamento fosse realizado pela aplicação, todos os registros recuperados pela consulta teriam que ser transferidos via rede para que fossem processados na estação cliente. Aplicações que tendem a processar um grande número de registros e apresentar apenas uma parte destes registros ao usuário final geram grande tráfego na rede [07].

- Aplicações que executam comandos SQL um por vez geram dois *trips* para cada comando executado. Se estes comandos estiverem agrupados em uma *stored procedure*, serão gerados apenas dois *trips* para todo o conjunto de comandos. Quanto maior o número de comandos agrupados em uma *stored procedure* menor será o tráfego gerado na rede [07].

- A utilização de *Stored procedures* em ambientes que tenham problemas com o tráfego da rede podem reduzir consultas extensas (grande quantidade de linhas) para apenas uma linha a ser transmitida pela rede [06].

Preservação de recursos da estação cliente:

- Aplicações que utilizam *stored procedures* transferem o processamento ao servidor, reduzindo a utilização de memória e acesso à disco na estação cliente [07].

Encapsulamento de privilégios:

- O desenvolvedor (ou DBA) necessita de privilégios de acesso/atualização na base de dados para a construção da *stored procedure*. Os usuários que utilizarão esta *stored procedure* não necessitarão destes privilégios explicitamente, pois já estarão encapsulados na *stored procedure* [07].

- Aos usuários pode ser dado o privilégio de execução de uma determinada *stored procedure* independentemente de terem permissão de acesso às tabelas utilizadas nesta rotina [06].

Redução de custos de desenvolvimento:

- Em um ambiente de aplicações para banco de dados muitas necessidades de processamento são repetitivas. Agrupando estas necessidades em *stored procedures* é possível compartilhar módulos de funções entre as aplicações (reutilização) [07].

- As *stored procedures* podem ser utilizadas por diversos usuários e aplicações. A utilização de maneira organizada garante a redução do ciclo de desenvolvimento com a reutilização de códigos [06].

Segurança, Administração e Manutenção Centralizada:

- Mantendo a codificação da aplicação em apenas um lugar (servidor), ocorre a simplificação das atividades de Segurança, Administração e Manutenção, pois não será necessário realizar o gerenciamento destas codificações em todas as estações que utilizam esta aplicação [07].

- As *stored procedures* viabilizam uma forma de compartilhamento de códigos pois inúmeros usuários e aplicações podem executar uma mesma *stored procedure* (reutilização de código). A manutenção se torna mais fácil pois a alteração em determinada *stored procedure* (realizada no servidor) é efetivada imediatamente para todos os usuários e aplicações [08].

Pré-compilação:

- O SGBD realiza a compilação da *stored procedure* apenas uma vez, reutilizando o plano de execução nas chamadas subseqüentes. Existe um ganho acentuado no desempenho de *stored procedures* chamadas repetidamente [06].

- No momento da criação da *stored procedure* já ocorre sua compilação (geração e otimização de um plano de execução). Ao ser executada pela primeira vez será transferida para a memória, ou seja, as próximas execuções utilizarão a *stored procedure* previamente compilada e disponível diretamente da memória. Comandos SQL não são tão eficientes quanto as *stored procedures* pois a cada execução precisam ser compilados e otimizados pelo SGBD (antes de serem executados efetivamente). Estas diferenças podem não ser perceptíveis em ambientes de menor escala ou mesmo nas estações que utilizam bancos de dados locais, entretanto, são altamente evidentes quando o ambiente de processamento engloba múltiplos bancos de dados e centenas de usuários concorrentes [08].

2.2.3 Exemplos

A *stored procedure* listada a seguir realiza o controle do número de sessões concorrentes, atualizando a tabela de histórico diário (mantendo sempre o maior valor detectado para o período):

```
procedure Audit_sess is
   nr_aux  number;
   cursor c1 is
      select sessions_current, sessions_warning,
         sessions_max, sessions_highwater
         from v$license;
begin
   for rReg in c1 loop
   begin
      select nr_current into nr_aux from audit_sessions
         where aa_referencia = to_number(to_char(sysdate,'YY'))
         and   mm_referencia = to_number(to_char(sysdate,'MM'))
         and   dd_referencia = to_number(to_char(sysdate,'DD'));
      if rReg.sessions_current > nr_aux then
         update audit_sessions
            set hr_referencia = to_number(to_char(sysdate,'HH24')),
            nr_current = rReg.sessions_current, nr_high = rReg.sessions_highwater
               where aa_referencia = to_number(to_char(sysdate,'YY'))
               and   mm_referencia = to_number(to_char(sysdate,'MM'))
               and   dd_referencia = to_number(to_char(sysdate,'DD'));
         commit;
      end if;
      exception
         when no_data_found then
            insert into audit_sessions
               values  (to_number(to_char(sysdate,'YY')),
                  to_number(to_char(sysdate,'MM')),
                  to_number(to_char(sysdate,'DD')),
                  to_number(to_char(sysdate,'HH24')),
                  rReg.sessions_current,
                  rReg.sessions_warning,
                  rReg.sessions_max,
                  rReg.sessions_highwater);
                  commit;
```

```
      end;
   end loop;
end audit_sess;
```

O exemplo abaixo mostra uma *stored procedure* denominada CREDITAR [09]. Esta *stored procedure* credita um valor para uma determinada conta corrente (ambas informações são passadas através de parâmetros).

```
CREATE PROCEDURE CREDITAR (nr_cc IN NUMBER, valor IN NUMBER) AS
   BEGIN
      UPDATE correntistas
      SET saldo = saldo + valor
      WHERE correntista_id = nr_cc;
   END;
```

Neste outro exemplo é demonstrada a chamada de um programa externo (codificado em linguagem C) que recebe o parâmetro informado à *stored procedure* [09].

```
CREATE PROCEDURE find_root (x IN REAL)
   IS LANGUAGE C
      NAME "c_find_root"
      LIBRARY c_utils
      PARAMETERS ( x BY REF );
```

O exemplo a seguir demonstra a chamada de comandos DDL (*Data definition language*) a partir de uma *stored procedure*.

```
CREATE OR REPLACE PROCEDURE altera_senha
   (usuario IN VARCHAR2, senha in varchar2) AS CID INTEGER;
BEGIN
    CID :=DBMS_SQL.OPEN_CURSOR;
    DBMS_SQL.PARSE(CID, 'alter user ' ||usuario|| ' identified by ' ||senha,
DBMS_SQL.NATIVE);
    DBMS_SQL.CLOSE_CURSOR(CID);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_SQL.CLOSE_CURSOR(CID);
END altera_senha;
```

A utilização de *stored procedures* alcança aplicações bastante diversificadas, sendo que os exemplos acima apenas ilustram algumas formas desta aplicabilidade. Outros exemplos também serão encontrados nos Anexos deste trabalho, bem como no capítulo que descreve a implementação do modelo proposto (baseado na utilização de *stored procedures*).

2.3 Modelos de Gerenciamento de Redes

Um dos modelos de gerenciamento adotados pela gerência de redes é o Modelo Internet [10] que é baseado na interação das entidades descritas abaixo:

Objeto gerenciado:

- É o recurso da rede que se pretende gerenciar (representação abstrata).

Gerente:

- É o processo que faz parte da aplicação distribuída sendo o responsável pelas atividades de gerenciamento (solicitação de operações aos agentes e recebimento de notificações dos mesmos).

Agente:

- É o processo que se encontra na outra ponta da aplicação distribuída. Responsável pela recepção das solicitações do gerente, execução de ações e envio de notificações ao processo gerente.

No Modelo Internet, através do SMI (*Structure of Management Information*), é realizada a definição das regras para descrição das informações de gerenciamento. Utilizando ASN.1 (*Abstract Syntax Notation* 1) são descritos os objetos gerenciados da MIB (*Management Information Base*) e também os pacotes que serão trocados pelo protocolo de gerenciamento. O protocolo utilizado, neste caso, é o SNMP (*Simple Network Management Protocol*).

O SNMP é o protocolo de gerenciamento (camada de aplicação da arquitetura TCP/IP) mais utilizado devido à sua simplicidade e facilidade de implementação. A primeira versão surgiu nos anos 80 mas existiam deficiências que foram corrigidas na segunda versão (1993), melhorando o suporte para a transferência de grandes blocos e implementando estratégias de gerenciamento centralizado. Uma terceira versão ainda foi lançada, focando especificamente o aspecto segurança. No gerenciamento SNMP é adicionado um agente (ao *hardware* ou ao *software*) que deverá realizar as coletas de dados dos dispositivos (armazenando-os na MIB). O gerente também é definido, normalmente através do desenvolvimento de um *software* com a finalidade de exibir as informações nas estações de monitoração da rede.

O SNMP é um protocolo de requisição/resposta tendo definidas quatro operações [10]:

- *Get*: Utilizada pelo gerente para a recuperação de uma instância de objeto do agente.

- *GetNext*: Utilizada pelo gerente para a recuperação da próxima instância de objeto do agente. Para a recuperação de todos elementos da lista do agente o

gerente emite uma operação *Get* inicial seguida de operações *GetNext* até o término da lista.

- *Set*: Utilizada pelo gerente para alterar valores de uma instância de objetos do agente.

- *Trap*: Utilizada pelo agente para informar o gerente (de forma assíncrona) a ocorrência de eventos.

Com o protocolo SNMP o gerente apenas obtém informações de um determinado equipamento. Para fazer frente à estas limitações surgiu a RMON MIB (*Remote Monitoring* MIB) que agregou a capacidade de gerenciamento remoto ao SNMP (os monitores do RMON trabalham de modo promíscuo, capturando todas as informações do tráfego da rede). As cinco funções implementadas pelos agentes que utilizam a RMON MIB são: Operação *off-line*, monitoração pró-ativa, detecção e registro de problemas, processamento sobre os dados coletados e múltiplos gerentes.

O modelo OSI [11] de gerenciamento de redes é, de certa forma, similar ao Modelo Internet. Ambos possuem pontos em comum, considerando-se os conceitos básicos de gerenciamento (gerente, agente e objetos gerenciados). O gerente é o responsável pelas solicitações ao agentes que executam operações sobre os objetos gerenciados (que são a representação dos recursos reais passíveis de gerenciamento). A base do sistema de gerenciamento também é a MIB.

Apesar de terem sido especificados uma série de protocolos de gerenciamento, para o Modelo Internet adotou-se o protocolo SNMP (devido ao seu maior desenvolvimento). No Modelo OSI destacou-se o protocolo CMIP (*Common Management Information Protocol*). Este protocolo é orientado à conexão e utiliza serviços do ACSE (*Association Control Service Element*), ROSE (*Remote Operations Service Element*) e também trabalha em conjunto com o CMIS (*Common Management Information Service*).

Como forma de interligar as duas arquiteturas de gerenciamento surgiu o protocolo CMOT (CMIP sobre TCP/IP). Este desenvolvimento permitiu a aplicação da estrutura de gerenciamento OSI sobre os objetos gerenciados de uma rede TCP/IP baseando-se nos modelos, serviços e protocolos da ISO.

2.4 Administração de bancos de dados

O administrador de banco de dados (DBA) é o responsável pela manutenção, segurança e performance dos servidores de bancos de dados. Suas atribuições incluem a monitoração do ambiente, manutenção das bases de dados, elaboração do planejamento de capacidade, medições e ajustes para a melhoria do desempenho das aplicações, definição e

implementação de políticas (segurança, *backup*, etc.) e outras atividades de suporte às equipes de desenvolvimento e usuários dos aplicativos em geral.

No desempenho destas atribuições, visando garantir o bom desempenho das aplicações que detêm as informações mais importantes da empresa, é importante que o DBA demonstre competência para garantir o funcionamento dos SGBD´s de forma otimizada, isenta de erros e com alta disponibilidade.

Para auxiliar o cumprimento destas metas, normalmente são utilizadas ferramentas auxiliares, ou seja, são adquiridos *softwares* de gerenciamento. Da mesma forma que os Modelos Internet e OSI, muitos destes *softwares* também implementam a relação agente/gerente, bem como bases de informação e protocolos de comunicação.

Nas pesquisas realizadas para a composição deste trabalho não foram encontrados modelos formais de gerenciamento para bases de dados SQL. Buscas de trabalhos acadêmicos e mesmo congressos de bancos de dados não permitiram relacionar nenhuma iniciativa que fosse semelhante ao modelo a ser proposto nesta tese.

O resultado destas pesquisas sempre convergiu para a identificação de produtos de gerenciamento disponíveis no mercado. Cada alternativa trazia suas particularidades e, normalmente, era focado para plataformas distintas, não tendo sido encontrado um único produto que pudesse ser utilizado para todos os SGBD´s SQL em geral.

Tendo em vista a ausência de um modelo formal e de um produto genérico para gerenciamento de bases de dados SQL, não considerou-se adequada a explanação detalhada da gama de produtos disponíveis no mercado. Ainda assim, optou-se por apresentar uma das alternativas existentes, pois será utilizada para a realização de uma análise comparativa com o modelo proposto. Esta descrição é o alvo do item a seguir.

2.5 Unicenter TNG

O mercado assistiu o surgimento de produtos de gerenciamento que possuem características que os distinguem: Utilização do SNMP para comandos de comunicação, *design* modular e API que permite a incorporação de funcionalidades através de módulos *plug-in* [12].

Conforme comentado anteriormente, não se pretende esgotar todas as soluções de mercado existentes, entretanto, uma destas alternativas será detalhada para permitir o comparativo com o modelo que será proposto neste trabalho.

Dentre a enorme gama de produtos (*softwares*) oferecidos pela CA (Computer Associates) existe o TNG (*The Next Generation*) que é composto por uma série de módulos. Os componentes principais do TNG são apresentados na figura 2.2 [13] [14]:

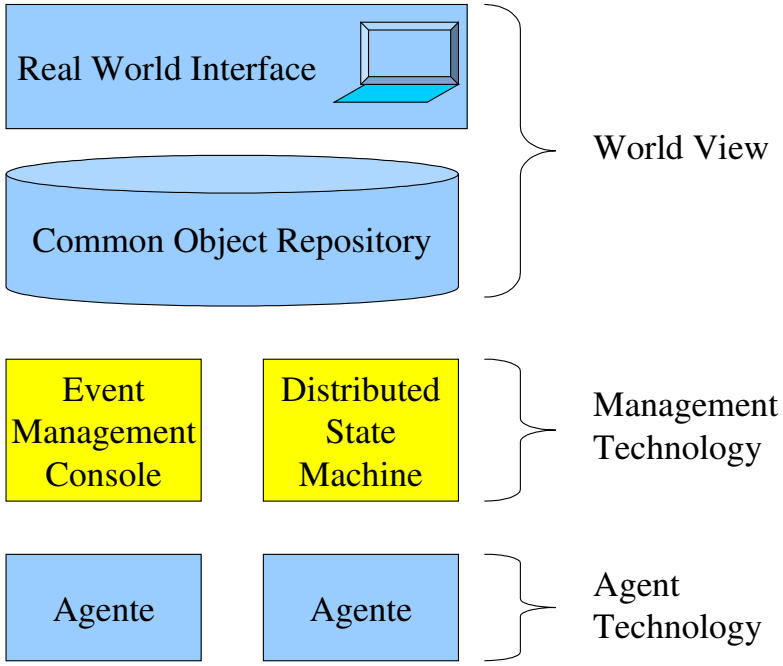

Figura 2.2 – Componentes do Unicenter TNG

O DSM (*Distributed State Machine*) exerce o papel de gerente . Para cada ambiente a ser monitorado são instalados os agentes específicos. Estes agentes realizam verificações (conforme especificações para cada recurso gerenciado) e enviam notificações (SNMP *traps*) quando da identificação de algum evento. Estes eventos são recebidos pela DSM e pela *Event Management Console* (Console de Eventos) que fará a tratativa para cada caso. Podem ser customizadas ações para determinadas notificações recebidas (como o envio de *e-mail* aos administradores).

No *World View* está contido o *Common Object Repository* que é o local onde estão armazenadas todas as funções de gerenciamento e os relacionamentos e propriedades dos recursos gerenciados. Estas informações são apresentadas nas estações monitoras pela *Real World Interface*, sendo que o Mapa 2-D é a alternativa mais utilizada. As estações monitoras podem ser utilizadas para realizar a verificação/alteração das especificações dos recursos gerenciados, bem como avaliar a situação geral dos ambientes (navegando pelos nós compostos pelo produto através do Mapa 2-D). A identificação da situação de cada nó é especificada através das cores verde, amarelo e vermelho (indicando ausência de alertas, avisos e alertas críticos, respectivamente). A explosão de cada nó permite a visualização de níveis inferiores, que por sua vez também podem ser explodidos até atingir o último nível

(recurso gerenciado). Neste ponto é permitido visualizar maiores detalhes sobre as causas dos alertas sinalizados para o recurso específico.

2.5.1 Agente Oracle

A forma de monitoramento realizada pelo agente Oracle do TNG se baseia na especificação de dois níveis para cada recurso gerenciado. Estes níveis são especificados na forma percentual. O primeiro nível (*warning*) define o valor que irá disparar um aviso e o segundo define o valor para a emissão de um alerta crítico.

Os recursos monitorados por este agente estão listados abaixo [15]:

- Pré-configurados no agente:
  - *Tablespaces*
  - *Data Disks*
  - *Data Files*
  - *Licensing* (*Users/Sessions*)

- Passíveis de serem configurados:
  - *Tables*
  - *Sequences*

A opção *Tablespaces* identifica aquelas que tenham atingido um determinado percentual de utilização (comparando o espaço total alocado e a área efetivamente em uso pelos objetos). Na verificação da fragmentação (*bubbles* e *honeycombs*) é apresentado o percentual de blocos livres (isolados e adjacentes) localizados entre os blocos de dados.

No caso dos *Data Disks* e *Data Files* podem ser especificados percentuais que definirão os níveis de criticidade para a realização de operações de I/O nos discos/volumes e nos *data files* (arquivos), respectivamente.

Através de *Licensing* (*Users* e *Sessions*) é realizado o controle de licenças. Usuários nominados são monitorados pela opção *users* enquanto que *sessions* verifica o número de sessões concorrentes (comparando com a quantidade de licenças informadas através de parâmetros do banco de dados).

Para *Tables* (tabelas) o percentual se aplica à quantidade de *extents* (extensões) que a tabela atingiu em relação ao máximo permitido.

Com a opção *Sequences* é realizada a verificação do intervalo (*range*) especificado para este tipo de objeto. O controle também é feito através de percentuais, visando identificar os ciclos que estão próximos de seus valores limítrofes (mínimos ou máximos).

Para utilizar a monitoração de *Tables* e *Sequences* é preciso incluir todos os objetos em que se pretenda efetuar este acompanhamento. As demais opções fazem o reconhecimento automático dos objetos existentes (utilizando valores *default* para os percentuais que identificarão os valores limites).

2.5.2 Demais agentes para bancos de dados SQL

Além do agente para bancos de dados Oracle, existem outros agentes disponibilizados pelo fabricante. Estes agentes, bem como os recursos monitorados por cada um estão listados a seguir:

DB2 [16]:

- Pré-configurados no agente:
  - nenhum

- Passíveis de serem configurados:
  - *Manager Information*
    - *memory status*
    - *sort heap*
  - *Database Information*
    - *backup status*
    - *cache status*
    - *SQL table status*
    - *connection information*
    - *counting*
    - *sort status*
    - *heap log*
    - *locking*
    - *integrity*
  - *Application Information*
    - *status*
    - *sort status*
    - *locks*
    - *SQL table status*
    - *cache status*
    - *cursor*
  - *Table Space Information*
    - *buffer writes*
    - *buffer index writes*
    - *I/O writes*
    - *I/O request*
    - *physical space*

Informix [17]:

- Pré-configurados no agente:
  - *Server Availability and Usage*
  - *Virtual Processor Utilization*
  - *Dbspace Capacity and Usage*
  - *Database Size*
  - *Database Table Size*
  - *Database Server Lock Events*
  - *Database Server Log Capacity and Usage*

Ingres [18]:

- Pré-configurados no agente:
  - DBMS *Servers*
  - *Communication Servers*
  - *General Communication Architecture* (GCA) *Layers*
  - *Cache* (*both local and shared*)
  - *Logging System*
  - *Locking System*

- Passíveis de serem configurados:
  - *Database tables*
  - *File Systems*

SQL Server [19]:

- Pré-configurados no agente:
  - *File Systems*
  - SQL Server *Processes*

- Passíveis de serem configurados:
  - SQL Server *Resource Usage*
  - *Databases*
  - *Transaction Logs*
  - SQL Server *Locks*
  - SQL Server *Tables*

Sybase [20]:

- Pré-configurados no agente:
  - *File Systems*
  - Sybase SQL Server *Processes*

- Passíveis de serem configurados:
  - Sybase SQL Server *Resource Usage*
  - *Databases*
  - *Transaction Logs*
  - Sybase SQL Server *Locks*
  - Sybase SQL Server *Tables*
  - *Devices*

Existe um agente específico para alguns SGBD´s (conforme lista apresentada) sendo que os recursos gerenciados variam conforme o agente utilizado, ou seja, não é adotada uma forma genérica para este gerenciamento (apesar de haverem alguns itens similares).

2.5.3 Unicenter TNG - SDK

Os tópicos anteriores visavam demonstrar que existem características particulares e formas diferenciadas de monitoração que não são atendidas pelos agentes de banco de dados do TNG (restritos aos recursos listados anteriormente). Conforme comentado, a CA disponibiliza uma gama considerável de produtos e poderia fornecer outros que poderiam suprir algumas destas deficiências constatadas. O intuito, contudo, era demonstrar que os produtos de gerenciamento, em menor ou maior grau, apresentam restrições quanto à sua utilização. Estas restrições podem ser decorrentes da própria concepção do produto ou mesmo porque existe uma característica muito particular em determinado ambiente e que não existe a possibilidade de customização deste item.

Dentre os produtos disponibilizados pelo fornecedor existe um que poderia ser utilizado para atenuar estas restrições. Trata-se do SDK (*Software Development Kit*) que permite a construção de aplicações com API´s (*Application Programming Interfaces*) para o Unicenter TNG [21] [22]. Este produto é direcionado a parceiros desenvolvedores e o seu principal objetivo é viabilizar a construção de novas aplicações integradas com o Unicenter TNG (sendo que esta característica é acenada como forte vantagem). Sem questionar o potencial deste produto, apenas deve-se salientar que, em última instância, o aplicativo desenvolvido é totalmente dependente do fornecedor e do produto que este disponibiliza.

2.6 Resumo

Neste capítulo foi apresentado o surgimento da linguagem SQL e dos SGBD´s relacionais baseados nesta linguagem. *Stored procedures* foram conceituadas, sendo que também foram apresentados os benefícios de sua utilização e alguns exemplos simples. O formato básico dos modelos de gerenciamento de redes (Internet e OSI) foi explanado. Foram abordados os aspectos de administração de bancos de dados, detalhando-se, também, o modelo adotado por uma das soluções de mercado. No próximo capítulo será apresentada uma proposta diferenciada para o gerenciamento de bancos de dados baseados na

linguagem SQL visando, principalmente, demonstrar uma alternativa que permita a definição e customização de quaisquer regras de gerenciamento desejadas.

3 Modelo Proposto

Este capítulo apresenta a proposta de solução para o problema de gerenciamento de bancos de dados SQL, detalhando sua composição, demonstrando a visão geral do modelo, suas características, o modelo de dados e suas principais vantagens.

3.1 Mapeamento do modelo proposto

Esta proposta adota o modelo genérico de gerenciamento de redes, empregando gerente(s), agentes, protocolo de comunicação e base de informação de gerenciamento. Neste modelo proposto, os objetos gerenciados são específicos para SGBD´s que utilizam o padrão SQL, de acordo com o escopo previamente definido para este trabalho. A figura 3.1 apresenta os componentes do modelo proposto.

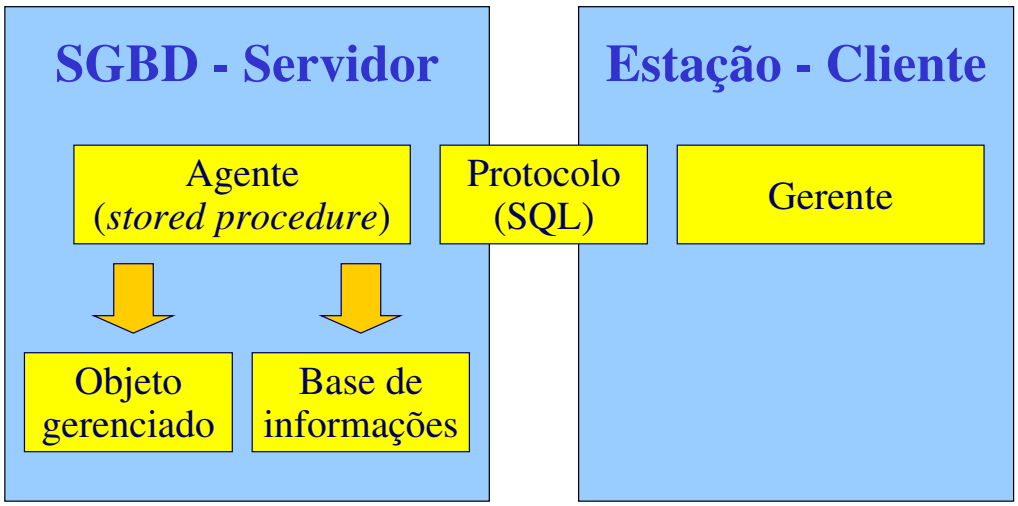

Figura 3.1 – Componentes do modelo proposto

3.1.1 Gerente

Em primeira instância o gerente deste modelo é o Administrador de Banco de Dados (gerente humano). Considerando-se que as regras de gerenciamento são customizáveis, a primeira responsabilidade do gerente será a definição dos objetos a serem gerenciados. Após esta definição deverá ser realizada a customização da regra de gerenciamento (criação da *stored procedure*) para cada objeto gerenciado, considerando-se as características e particularidades de cada objeto. Definida a regra de gerenciamento caberá, ainda, ao gerente o cadastramento e agendamento da execução periódica da mesma. Todas estas ações são executadas através de ferramentas disponibilizadas por cada SGBD utilizando-se a linguagem SQL.

Através da técnica de *polling* o gerente solicitará informações sobre os objetos gerenciados, recebendo informações sobre alertas emitidos pelas regras customizadas de cada objeto. O conhecimento do Administrador de Banco de Dados será utilizado para a resolução de problemas, ajustes no ambiente e programação de ações corretivas automáticas. Estas ações corretivas são criadas dentro da própria regra customizada (*stored procedure*), ou seja, a regra de emissão de alerta também ficará responsável pela correção do problema identificado (quando for o caso).

No modelo proposto é possível a utilização de múltiplos gerentes, permitindo o compartilhamento ou distribuição de responsabilidades sobre determinadas instâncias entre os Administradores de Bancos de Dados (ou mesmo para as áreas de Operação e *Help Desk*). Grupos de instâncias podem ser criados para realizar esta distribuição. Cada gerente poderá definir vários grupos, bem como criar um grupo único de gerenciamento, de acordo com sua necessidade.

Além das notificações de alertas, o gerente também receberá informações sobre o(s) grupo(s) de instâncias gerenciadas. Esta notificação consiste na elaboração de um resumo de cada grupo, informando as condições do módulo de gerenciamento e o estado das instâncias gerenciadas. A definição do grupo de instâncias, o gerenciamento (monitoração) destas instâncias e o controle da situação geral do módulo de gerenciamento são realizados através de *script* provido pelo modelo proposto.

3.1.2 Agente

Tradicionalmente este componente é adicionado ao objeto gerenciado, realizando o seu controle. No modelo proposto, estes componentes serão adicionados ao próprio SGBD. Entretanto, não será agregado a cada objeto do banco de dados a ser gerenciado, pois os agentes serão configurado através da criação de *stored procedures*.

Um objeto gerenciado poderá ter uma ou mais *stored* procedures definidas para o seu gerenciamento. No modelo proposto não optou-se pela definição de uma única *stored procedure* (regra customizada) por objeto gerenciado. Esta determinação permite a definição de critérios diversos para cada objeto, adotando, inclusive, agendamentos de execuções distintos para cada caso. A própria composição do modelo proposto já prevê a adoção de *stored procedures* com funções distintas para o mesmo objeto (coleta de dados históricos e emissão de alertas, por exemplo).

O agente, definido através da *stored procedure*, incorpora outra função específica (além da coleta de dados históricos e emissão de alertas). Trata-se da ação corretiva automática que, neste caso, estará presente na própria regra de emissão de alerta. O Administrador de Banco de Dados, identificando uma situação crítica para determinado objeto, poderá definir esta ação corretiva (se julgar necessário). Determinadas regras poderão ser definidas apenas para a emissão de alertas e, posteriormente, serem ajustadas

para incorporar uma ação corretiva (a flexibilidade do modelo proposto é uma de suas principais vantagens).

Os dados coletadas pelo agente, bem como as próprias informações de controle de execução são atualizadas em tabelas do banco de dados. No modelo proposto existem três *stored procedures* pré-definidas, a saber: Sumarização, Expurgo e Verificação. Estas *stored procedures* também são consideradas agentes, pois realizam operações de controle nos dados históricos dos objetos gerenciados (Sumarização e Expurgo) e envio de informações ao gerente (Verificação). Estes agentes apenas diferem dos demais (criados conforme as necessidades de gerenciamento) por já fazerem parte do modelo proposto e terem sido definidos de forma genérica, ou seja, atendem todos os objetos gerenciados que forem incorporados ao módulo de gerenciamento.

3.1.3 Protocolo de comunicação

Conforme comentado anteriormente, o modelo proposto ficará inserido no próprio SGBD. Esta característica, aliada a composição de *stored procedures* realizando o papel de agentes, dispensa a necessidade de requisições específicas do gerente para o agente.

A única requisição do gerente para o agente (realizada através de *polling*) é feita através da utilização de *script* de monitoração (residente no gerente) e execução da *stored procedure* de Verificação (residente no SGBD – agente). Demais intervenções do gerente no módulo de gerenciamento são realizadas através da utilização da linguagem SQL em ferramentas disponibilizadas por cada SGBD, não tendo envolvimento do agente. Dentre estas ações destacam-se a criação das *stored procedures*, cadastramento dos objetos gerenciados, definição de parâmetros de sumarização e expurgo, bem como verificação detalhada de alertas e históricos dos objetos gerenciados.

Atualizações de valores dos objetos gerenciados (coletas de informações e emissões de alertas) são realizadas pelo agente na própria base de dados (tabelas). Mesmo as ações corretivas automáticas são executadas independentemente do gerente. Neste caso, é o gerente que as define e configura, entretanto, ficam incorporadas na *stored procedure* (agente) e executam quando da identificação de uma situação crítica (o gerente apenas será notificado que o alerta ocorreu e que a ação corretiva foi executada).

3.1.4 Base de informação de gerenciamento (MIB)

Como as regras customizadas são compostas através da elaboração de *stored procedures*, serão utilizadas tabelas do próprio SGBD para efetuar o armazenamento de dados. Contudo, desenvolveu-se a SQL-MIB que engloba todas as entidades descritas no modelo de dados (item posterior deste capítulo) para formalizar a apresentação destas entidades (adotando-se o modelo de informação da arquitetura de gerenciamento Internet).

```
SQL-MIB DEFINITIONS ::= BEGIN

IMPORTS
        OBJECT-TYPE, MODULE-IDENTITY, experimental, Integer32
                FROM SNMPv2-SMI
        DateAndTime, DisplayString
                FROM SNMPv2-TC
        MODULE-COMPLIANCE, OBJECT-GROUP
                FROM SNMPv2-CONF

-- SQL-MIB

sqlMib MODULE-IDENTITY
LAST-UPDATED    "0212041200Z"
ORGANIZATION    "UFSC-CPGCC"
CONTACT-INFO
        "Guilherme Huth
        Rua João Pessoa, 2021
        Costa e Silva - Joinville - SC
        89218-533 - Brazil
        Email: guilhh1@yahoo.com.br"
DESCRIPTION
        "The MIB module to describe objects for a generic management model for SQL
        databases."
::= { experimental 9999 }

--
-- Registration hierarchy for this MIB
--

sqlManag                OBJECT IDENTIFIER ::= { sqlMib 1 }
sqlHistory              OBJECT IDENTIFIER ::= { sqlMib 2 }
sqlAlert                OBJECT IDENTIFIER ::= { sqlMib 3 }
sqlMonitor              OBJECT IDENTIFIER ::= { sqlMib 4 }
sqlConformance          OBJECT IDENTIFIER ::= { sqlMib 5 }

-- SQL MIB

-- Management Object

sqlManagTable OBJECT-TYPE
        SYNTAX          SEQUENCE OF SqlManagEntry
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "This table holds all Stored procedures created to management
                objects."
        ::= { sqlManag 1}

sqlManagEntry OBJECT-TYPE
        SYNTAX          SqlManagEntry
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "An entry in the sqlManagTable to define management characteristics."
        INDEX { sqlManagId }
        ::= { sqlManagTable 1 }

SqlManagEntry ::=
        SEQUENCE {
                sqlManagId              Integer32,
                sqlManagType            DisplayString,
                sqlManagName            DisplayString,
                sqlManagBegExec         DateAndTime,
                sqlManagEndExec         DateAndTime,
                sqlManagSumPeriod       DisplayString,
                sqlManagSumType         DisplayString,
                sqlManagRetPeriod       DisplayString,
                sqlManagRetQty          Integer32
                }
```

```
sqlManagId OBJECT-TYPE
        SYNTAX          Integer32 (1..9999)
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "An index that uniquely identifies a management."
        ::= { sqlManagEntry 1 }

sqlManagType OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The management's type can be (H)istory and (A)lert. There are also
              special types for two procedures provided by the model: (S)ummarization
              and (E)limination."
        ::= { sqlManagEntry 2 }

sqlManagName OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The name of the stored procedure that colects data history or
              generates alert(s)."
        ::= { sqlManagEntry 3 }

sqlManagBegExec OBJECT-TYPE
        SYNTAX          DateAndTime
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "The begin date/time of the stored procedure last execution."
        ::= { sqlManagEntry 4 }

sqlManagEndExec OBJECT-TYPE
        SYNTAX          DateAndTime
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "The end date/time of the stored procedure last execution."
        ::= { sqlManagEntry 5 }

sqlManagSumPeriod OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The summarization's period can be H, D, S, M and A for hour, day,
              week, month and year, respectively."
        ::= { sqlManagEntry 6 }

sqlManagSumType OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The summarization's type can be U, I, and A for last, minimum and
              maximum, respectively."
        ::= { sqlManagEntry 7 }

sqlManagRetPeriod OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The retention's period can be D, S, M and A for day, week, month, and
              year, respectively."
        ::= { sqlManagEntry 8 }

sqlManagRetQty OBJECT-TYPE
        SYNTAX          Integer32
```

```
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION
              "The  retention's  quantity  defines  the  number  of  periods  for  data
        retention."
 ::= { sqlManagEntry 9 }

-- History Object

sqlHistoryTable OBJECT-TYPE
        SYNTAX         SEQUENCE OF SqlHistoryEntry
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION
               "This table holds all history data for management objects."
        ::= { sqlHistory 1}

sqlHistoryEntry OBJECT-TYPE
        SYNTAX         SqlHistoryEntry
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION
               "Entries in the sqlHistoryTable to keep history information for
               management objects."
        INDEX { sqlHistoryId, sqlHistoryDate, sqlHistorySupplDesc }
        ::= { sqlHistoryTable 1 }

SqlHistoryEntry ::=
        SEQUENCE {
               sqlHistoryId          Integer32,
               sqlHistoryDate        DateAndTime,
               sqlHistorySupplDesc   DisplayString,
               sqlHistoryData        DisplayString,
               sqlHistorySum         DisplayString
               }

sqlHistoryId OBJECT-TYPE
        SYNTAX         Integer32 (1..9999)
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION
              "The identification that relates the data history to the
              sqlManagTable."
        ::= { sqlHistoryEntry 1 }

sqlHistoryDate OBJECT-TYPE
        SYNTAX         DateAndTime
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION
               "The date/time when the data history was collected."
        ::= { sqlHistoryEntry 2 }

sqlHistorySupplDesc OBJECT-TYPE
        SYNTAX         DisplayString
        MAX-ACCESS             read-create
        STATUS         current
        DESCRIPTION
              "A supplementar description is used for multiple entries in the same
              collection."
        ::= { sqlHistoryEntry 3 }

sqlHistoryData OBJECT-TYPE
        SYNTAX         DisplayString
        MAX-ACCESS     read-create
        STATUS         current
        DESCRIPTION
              "All data history for the object. That data can be a single value (for
              any type) or a concatenation of values."
        ::= { sqlHistoryEntry 4 }

sqlHistorySum OBJECT-TYPE
```

```
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The summarization's flag can be S or N, indicating a summarized value
              or not, respectively."
        ::= { sqlHistoryEntry 5 }

-- Alert Object

sqlAlertTable OBJECT-TYPE
        SYNTAX          SEQUENCE OF SqlAlertEntry
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "This table holds all alerts generated for management objects."
        ::= { sqlAlert 1}

sqlAlertEntry OBJECT-TYPE
        SYNTAX          SqlAlertEntry
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
               "Entries in the sqlAlertTable to keep alerts generated for management
               objects."
        INDEX { sqlAlertId, sqlAlertDate, sqlAlertSupplDesc }
        ::= { sqlAlertTable 1 }

SqlAlertEntry ::=
        SEQUENCE {
               sqlAlertId            Integer32,
               sqlAlertDate          DateAndTime,
               sqlAlertSupplDesc     DisplayString,
               sqlAlertData          DisplayString,
               sqlAlertDesc          DisplayString,
               sqlAlertSum           DisplayString
               }

sqlAlertId OBJECT-TYPE
        SYNTAX          Integer32 (1..9999)
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The identification that relates the alert to the sqlManagTable."
        ::= { sqlAlertEntry 1 }

sqlAlertDate OBJECT-TYPE
        SYNTAX          DateAndTime
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
               "The date/time when the alert was generated."
        ::= { sqlAlertEntry 2 }

sqlAlertSupplDesc OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "A supplementar description is used for multiple entries in the same
              alert generation."
        ::= { sqlAlertEntry 3 }

sqlAlertData OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The value(s) for the object that arises a alert situation."
        ::= { sqlAlertEntry 4 }

sqlAlertDesc OBJECT-TYPE
```

```
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The explanation of the cause of the alert."
        ::= { sqlAlertEntry 5 }


sqlAlertSum OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
              "The summarization's flag can be S or N, indicating a summarized value
              or not, respectively."
        ::= { sqlAlertEntry 6 }

-- Monitor Object

sqlMonitorTable OBJECT-TYPE
        SYNTAX          SEQUENCE OF SqlMonitorEntry
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "This table holds all station that are monitoring SQL database(s)."
        ::= { sqlMonitor 1}

sqlMonitorEntry OBJECT-TYPE
        SYNTAX          SqlMonitorEntry
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "Entries in the sqlMonitorTable are generated automatically when any
                station connect to the database. Old entries are automatically
                eliminated after 72 hours without activity."
        INDEX { sqlMonitorId }
        ::= { sqlMonitorTable 1 }

SqlMonitorEntry ::=
        SEQUENCE {
                sqlMonitorId          DisplayString,
                sqlMonitorAlertDate   DateAndTime,
                sqlMonitorManagDate   DateAndTime
                }

sqlMonitorId OBJECT-TYPE
        SYNTAX          DisplayString
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "An index that uniquely identifies a monitor."
        ::= { sqlMonitorEntry 1 }

sqlMonitorAlertDate OBJECT-TYPE
        SYNTAX          DateAndTime
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "The date/time of the last alert verification."
        ::= { sqlMonitorEntry 2 }

sqlMonitorManagDate OBJECT-TYPE
        SYNTAX          DateAndTime
        MAX-ACCESS      read-create
        STATUS          current
        DESCRIPTION
                "The date/time of the last summary report sent to the monitor."
        ::= { sqlMonitorEntry 3 }

-- Conformance Information

sqlCompliances          OBJECT IDENTIFIER ::= { sqlConformance 1 }
```

```
sqlGroups               OBJECT IDENTIFIER ::= { sqlConformance 2 }

-- Compliance Statements

sqlCompliance MODULE-COMPLIANCE
         STATUS         current
         DESCRIPTION
                "Describes the requirements for conformance to the sql MIB."
         MODULE --this module
                MANDATORY-GROUPS {    sqlManagGroup,
                                      sqlHistoryGroup,
                                      sqlAlertGroup,
                                      sqlMonitorGroup }

         ::= { sqlCompliances 1 }

-- Units of Conformance

sqlManagGroup OBJECT-GROUP
         OBJECTS {      SqlManagType,
                        SqlManagName,
                        SqlManagBegExec,
                        SqlManagEndExec,
                        SqlManagSumPeriod,
                        sqlManagSumType,
                        sqlManagRetPeriod,
                        sqlManagRetQty }
         STATUS         current
         DESCRIPTION
                "The Management Group."
         ::= { sqlGroups 1 }

sqlHistoryGroup OBJECT-GROUP
         OBJECTS {      sqlHistoryData,
                        sqlHistorySum }
         STATUS         current
         DESCRIPTION
                "The History Group."
        ::= { sqlGroups 2 }

sqlAlertGroup OBJECT-GROUP
         OBJECTS {      SqlAlertData,
                        SqlAlertDesc,
                        sqlAlertSum }
         STATUS         current
         DESCRIPTION
                "The Alert Group."
         ::= { sqlGroups 3 }

sqlMonitorGroup OBJECT-GROUP
         OBJECTS {      sqlMonitorAlertDate,
                        sqlMonitorManagDate }
         STATUS         current
         DESCRIPTION
                "The Monitor Group."
         ::= { sqlGroups 4 }

END
```

3.2 Visão Geral

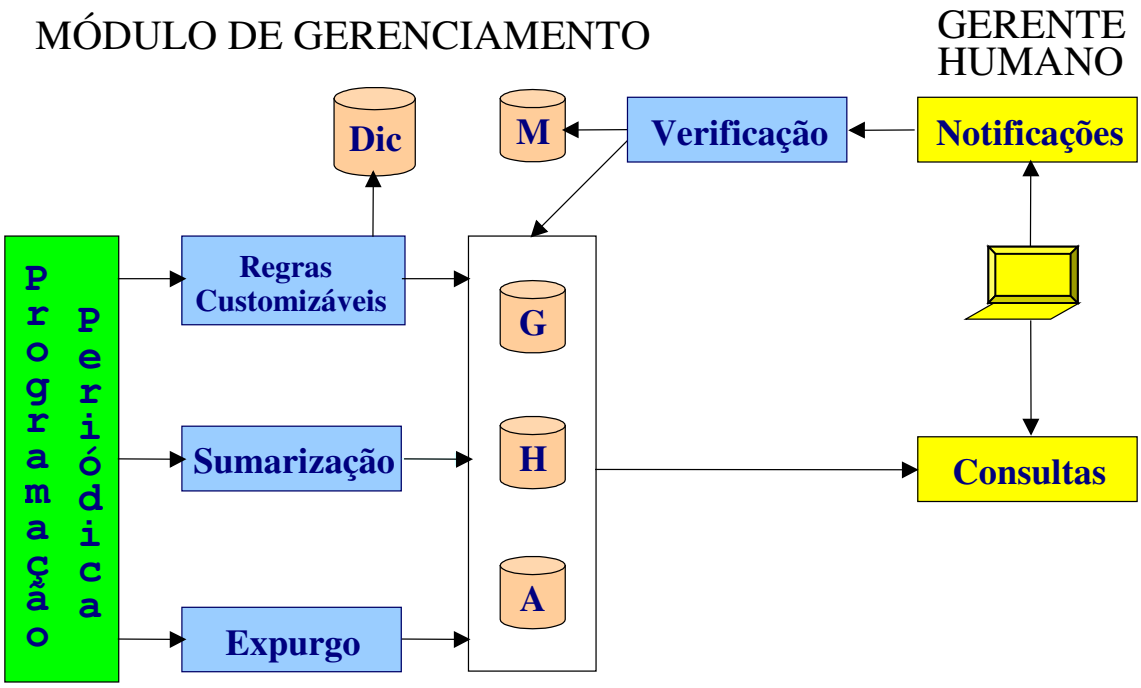

Figura 3.2 – Visão geral do modelo proposto

Existem três entidades centrais que sustentam o modelo proposto. Tratam-se das tabelas Gerenciamento (G), Histórico (H) e Alerta (A). A tabela Gerenciamento é responsável pelo armazenamento (cadastro) das regras que serão criadas/customizadas. A tabela Histórico é o repositório genérico para armazenamento dos dados coletados pelas regras. Estas informações serão utilizadas por outras regras customizadas (realização de projeções sobre os objetos gerenciados). A tabela Alerta destina-se ao armazenamento das mensagens de aviso/erro em objetos gerenciados (identificados pelas regras).

As regras customizáveis (*stored procedures*) são as responsáveis pela gravação dos dados históricos (H) e geração dos alertas (A) para os objetos gerenciados. Estas regras consultam tabelas/visões do próprio SGBD (Dic) e/ou dados históricos (H).

As funções de Sumarização e Expurgo aplicam-se às tabelas de dados históricos (H) e alertas (A), efetuando o controle do volume de dados armazenados.

Tanto as regras customizáveis como as funções de Sumarização e Expurgo tem o controle de execução efetuado pela Programação Periódica.

A descrição acima sintetiza o módulo de gerenciamento incorporado ao próprio SGBD. Os demais componentes referem-se a figura do Gerente Humano que complementa o modelo proposto, realizando consultas nas tabelas de dados históricos e alertas. Além disso, realiza também a execução da função de Verificação que utiliza uma tabela para o cadastramento das estações de monitoração (M). Esta verificação consiste em receber notificações para novos alertas e também o resumo do *status* de gerenciamento das instâncias que estão sob o seu controle (garantindo que o módulo de gerenciamento está ativo e funcionamento adequadamente).

3.3 Características básicas

3.3.1 Programação periódica

Este característica visa suprir a execução dos processos de gerenciamento de acordo com a periodicidade previamente definida. Determinados bancos de dados SQL apresentam a possibilidade de definição de tarefas periódicas. No caso da ausência desta capacidade será adotado o mecanismo existente no sistema operacional. A programação periódica está encarregada de verificar, disparar e registrar a execução das *stored procedures* de gerenciamento (além das funções de sumarização e expurgo – ver próximos itens).

A principal vantagem desta característica é a automatização e o controle da execução destes processos. A ausência da programação periódica das tarefas de gerenciamento tornaria o modelo inviável, pois demandaria a intervenção manual constante do gerente humano (descaracterizando o modelo de gerenciamento proposto).

3.3.2 Regras Customizáveis

A adequação desta característica é realizada através da utilização de *stored procedures*, agregando ao modelo extrema facilidade para a configuração de qualquer tipo de verificação (e/ou registro). Através da linguagem SQL é possível definir critérios que atendam ilimitadas situações e/ou implementem a verificação de qualquer condição de determinado objeto gerenciado, ou seja, customização de regras sem restrições.

Neste modelo as necessidades específicas são atendidas de forma rápida e eficiente, permitindo, inclusive, a implementação de regras baseadas em critérios de negócio. As regras são definidas através de *stored procedures* (SQL), havendo uma rotina para cada tipo de gerenciamento requerido. Pode haver mais de uma *stored procedure* para um objeto. Cada s*tored procedure* realiza as verificações pertinentes ao seu objeto gerenciado e armazena as informações (alerta e histórico) em tabelas do banco de dados e/ou arquivos do sistema operacional.

As *stored procedures* poderão ser definidas para atender finalidades distintas, a saber: emissão de alertas baseados na situação atual, coleta de dados históricos e emissão de alertas baseados nos dados históricos.

A emissão de alertas baseados na situação atual não depende de informações históricas. Caracterizam-se por situações passíveis de serem verificadas através de consultas à visões e/ou tabelas pertencentes ao SGBD. Nestes casos se enquadram verificações de valores cujos limites já são previamente conhecidos. Exemplo: quantidade de usuários conectados ao SGBD não pode ultrapassar o valor especificado através de parâmetro do banco de dados.

A coleta de dados históricos destina-se ao armazenamento de informações sobre determinado objeto. Esta coleta ocorre periodicamente e não realiza nenhuma consistência ou verificação para a emissão de alertas. Estes dados servirão para os casos em que apenas a situação atual não seja suficiente para a constatação de uma necessidade de alerta ao administrador. A realização de coletas freqüentes durante um longo período de tempo e a emissão de um único alerta para todo o período também pode ser atendida através da coleta de dados históricos. Neste caso seriam evitadas as emissões de inúmeros alertas (no caso de apenas ser desejado a identificação do valor mais crítico do período). Exemplo: necessidade de coletar (ao longo do dia) o número de cursores em aberto com emissão de alerta ao final do dia caso o valor máximo tenha atingido 90% do limite (especificado através de parâmetro do banco de dados).

A emissão de alertas baseados nos dados históricos visa avaliar o comportamento ao longo do tempo de determinado objeto, não restringindo-se apenas à situação atual do mesmo. Através de uma tendência observada nos dados históricos poderá ser identificada antecipadamente uma situação crítica. Exemplo: a utilização da capacidade de determinado objeto encontra-se em apenas 70%, mas no decorrer de uma hora já atingiu 80%. Com estas duas coletas pode-se estimar que este objeto atingirá 100% de utilização nas próximas duas horas. Se estas coletas fossem feitas baseando-se apenas na situação atual provavelmente não seria identificada uma situação de anormalidade (pois 70% e mesmo 80% não são percentuais elevados de utilização).

As regras podem receber os valores referenciais através de parâmetros. Com esta característica o modelo adquire maior flexibilidade. No caso de algum valor referencial ficar defasado, basta informar o novo valor na chamada da *stored procedure* (Programação Periódica). Após este ajuste, as próximas verificações de alertas já estarão realizando o comparativo baseadas no novo valor (não existe necessidade de alterar a codificação da regra).

Necessidades esporádicas (simulações) também podem ser realizadas através de execuções das regras com valores diferenciados. Neste caso bastaria executar a *stored procedure* desejada, informando o(s) valore(s) referenciais desejado(s). Também podem

ocorrer situações ou períodos em que os valores referenciais precisem ser ajustados. Nestas situações, é possível manter a programação normal e acrescentar outras programações para estas necessidades específicas.

Outras aplicações também podem ser adequadas com a passagem de parâmetros (exemplo: quantidade de dias a serem pesquisados na base de dados históricos). Neste caso, pode ocorrer de uma regra receber mais do que um parâmetro. Regras que utilizem critérios mais elaborados (composição de vários valores referenciais), também seriam casos de *stored procedures* que utilizariam dois ou mais parâmetros para validação de algum objeto gerenciado.

Formato geral das regras customizáveis

Declaração
Identificação da regra (nome da *stored procedure*)
Parâmetros de entrada (valores referenciais)
Variáveis de trabalho e cursores

Execução	**Alerta**	**Histórico**
Leitura da data/hora última execução (se regra referenciada)	X	
Atualização data/hora início última execução	X	X
Coleta do(s) valor(es) desejado(s)		X
Gravação do(s) dado(s) histórico(s)		X
Verificação dos valores (situação atual ou dados históricos)	X	
Gravação do(s) alerta(s) (caso necessário)	X	
Ação corretiva (se for o caso)	X	
Atualização data/hora término última execução	X	X

3.3.3 Histórico Genérico

A estrutura de armazenamento comporta todos os tipos de regras que eventualmente sejam implementados. Esta característica é mandatória, pois sua inexistência tornaria o modelo inflexível (exigindo uma estrutura específica para cada regra implementada). A utilização de uma base genérica garante o armazenamento dos históricos em um local único, independentemente da quantidade e do tipo das regras implementadas. Esta característica mantém a facilidade na definição de regras (dispensando a definição de bases adicionais para o armazenamento dos dados coletados pela regra). Os dados históricos servirão para auxiliar a tomada de decisão de determinados objetos gerenciados e também permitirão a extração de relatórios de acompanhamento (planejamento de capacidade, por exemplo).

A utilização de uma estrutura genérica para os dados históricos comporta diferentes tipos de dados a serem armazenados. Como não existem restrições para as definições de

regras, haverão situações em que números, datas e/ou caracteres deverão ser armazenados utilizando-se o mesmo campo da tabela de dados históricos. O modelo proposto adota um campo único do tipo caracter (de tamanho variável). Este tipo de campo atende a necessidade de armazenamento de um grande volume de informações, mas não desperdiça espaço pois tem tamanho variável (apenas posições que contenham dados serão armazenadas). Através das funções de conversão disponíveis na linguagem SQL é possível utilizar o campo do tipo caracter para armazenar números ou datas (ou a combinação de vários tipos). O retorno dos valores armazenados ao seu formato original também será realizado por funções da linguagem SQL.

A utilização deste tipo de campo também permite o armazenamento de mais do que um valor para a coleta realizada. Esta característica viabiliza agrupar mais do que um valor, com tamanhos e formatos diferentes, em um campo único, ou seja, utilizando o mesmo registro. Neste caso deve-se observar, apenas, que o valor mais significativo ocupe as primeiras posições do campo composto. Esta recomendação é válida para os casos de sumarização (ver próximo item) que utilizem como critério a retenção dos valores mínimos ou máximos.

As regras que realizam as coletas de dados históricos poderão armazenar entradas unitárias ou múltiplas para cada objeto gerenciado. As entradas unitárias ocorrerão para um objeto gerenciado que esteja associado à somente um valor recuperado. Exemplo: uma regra realizará a coleta do número total de conexões concorrentes no banco de dados. Este tipo de coleta representará uma entrada unitária, ou seja, somente um registro coletado à cada execução da regra.

Existem outros casos, entretanto, que poderão armazenar entradas múltiplas para o mesmo objeto gerenciado. Exemplo: uma regra realizará a coleta do número de conexões concorrentes de cada usuário do banco de dados. Para estas situações existe uma descrição complementar na estrutura de dados históricos (não utilizada para entradas unitárias). Neste exemplo, o nome de cada usuário será armazenado na descrição complementar (garantindo a unicidade destas entradas múltiplas para o mesmo objeto gerenciado).

No caso de coletas que gerem múltiplas entradas deve-se utilizar o mesmo valor para a data/hora do histórico de todas as entradas. Este requisito é necessário para garantir o agrupamento destas coletas. Exemplo: podem existir regras de verificação dos dados históricos que selecionam a última ocorrência. A utilização de valores diferentes para data/hora de cada entrada (do mesmo lote) faria com que apenas a última entrada fosse recuperada (quando o desejado seria a recuperação de todos os registros do lote).

3.3.4 Sumarização

Esta característica se aplica ao repositório dos dados históricos e alertas. O objetivo da sumarização é preservar apenas o registro mais significativo dentro de determinado

período (conforme configuração do objeto gerenciado). Para trazer maior acuracidade aos dados, poderão haver coletas freqüentes para determinados objetos. O armazenamento de todas as coletas realizadas, entretanto, pode representar uma quantidade elevada de entradas na tabela de dados históricos. Por exemplo, poderão haver coletas realizadas a cada 5 minutos, mas a opção de sumarização permitirá que seja armazenado ao final do dia apenas um registro correspondente ao valor máximo encontrado nas coletas. Os períodos de sumarização, bem como a computação do valor podem ser customizados conforme a necessidade ou característica do objeto gerenciado.

A implementação da função de sumarização é feita através de uma *stored procedure*, sendo que as execuções desta função são atendidas pela "Programação Periódica" (característica mencionada anteriormente). As cláusulas válidas para os critérios de sumarização são "mínimo", "máximo", e "último". Conforme mencionado no item anterior, no caso de valores compostos, é recomendado que as primeiras posições do campo sejam ocupadas pelo valor mais significativo (quando utilizadas as cláusulas "mínimo" e "máximo").

Existem algumas considerações sobre a não utilização da média para a sumarização dos dados. A *stored procedure* de sumarização é genérica (atende todos os dados históricos e alertas armazenados). Enquanto que as cláusulas "mínimo", "máximo" e "último" podem ser aplicadas pela *stored procedure* diretamente sobre o campo armazenado, o cálculo da média não seria possível. Seria necessário converter o dado armazenado ao seu formato original (numérico). Para realizar esta conversão seria preciso adotar um tamanho único para todos os valores numéricos a serem armazenados ou definir várias funções de sumarização (dependendo do tamanho dos valores armazenados). Ambas alternativas trazem desvantagens, seja desperdiçando espaço de armazenamento ou descaracterizando o modelo genérico proposto.

Estas considerações, entretanto, não sentenciam a impossibilidade de verificação de valores médios nas regras de gerenciamento. A composição de regras customizáveis (sem limitações) é a principal característica deste modelo. A sumarização serve para otimizar o armazenamento dos dados históricos (sem perda da acuracidade) e alertas. A composição de regras que calculem os valores médios sobre os dados históricos é plenamente possível e podem ser utilizadas sem restrições (independentemente da forma de sumarização adotada).

A tabela 3.1 demonstra o percentual de registros históricos mantidos, utilizando-se a sumarização, considerando-se o período de retenção anual. Se for considerada uma única coleta diária, tem-se 365 registros ao final do primeiro ano. Esta quantidade seria reduzida para 52, 12 e 1 (registros) utilizando-se a sumarização por semana, mês e ano, respectivamente. Aumentando-se o número de coletas diárias para 24 (hora em hora) seriam totalizados 8.760 registros ao ano (sem sumarização). Utilizando-se a sumarização, a quantidade de registros seria reduzida para 365, 52, 12 ou 1 (por dia, semana, mês ou ano,

respectivamente). Desta forma, o percentual de registros armazenados reduziria consideravelmente.

Tabela 3.1 – Percentual de registros históricos mantidos com a sumarização.

Registros coletados		**% de registros mantidos com a sumarização**			
Dia	**Ano**	**D(iária)**	**S(emanal)**	**M(ensal)**	**A(nual)**
1	365	100,00	14,25	3,29	0,27
3	1095	33,33	4,75	1,10	0,09
5	1825	20,00	2,85	0,66	0,05
10	3650	10,00	1,42	0,33	0,03
12	4380	8,33	1,19	0,27	0,02
24	8760	4,17	0,59	0,14	0,01

A opção de sumarização permite a definição de coletas freqüentes e uma forma econômica de armazenamento (sem perda da acuracidade dos dados históricos). No caso dos alertas também existe a sumarização, entretanto, não é possível realizar estimativa semelhante à que foi demonstrada para os dados históricos, pois a quantidade de alertas emitidos irá depender das situações reais defrontadas (independentemente da programação de verificação). Ainda assim é possível afirmar que a sumarização garantirá a redução de registros armazenados (mantendo-se o registro mais significativo dos alertas emitidos para o período especificado).

Formato geral da *stored procedure* Sumarização

Declaração
Identificação da *stored procedure*: Sumarizacao
Parâmetro de entrada (Indicador do período de sumarização)
Variáveis de trabalho e cursores Histórico e Alerta

Execução
Atualização data/hora início última execução
Formatação da data de referência para sumarização conforme parâmetro
Seleção dos registros históricos (preservando o mais significativo)
Eliminação dos demais registros históricos correspondentes ao período sumarizado
Seleção dos registros de alerta (preservando o mais significativo)
Eliminação dos demais registros de alerta correspondentes ao período sumarizado
Atualização data/hora término última execução

3.3.5 Expurgo

Característica também aplicada ao repositório dos dados históricos e alertas. Além das especificações do período e forma de sumarização, cada gerenciamento também terá,

obrigatoriamente, a definição do período de retenção. Dados que atingirem a data de retenção especificada são automaticamente eliminados da base de dados. Esta característica visa confrontar uma situação bastante comum nas aplicações implementadas, ou seja, evitar que o início da definição do processo de expurgo ocorra apenas quando já houver uma quantidade elevada de dados armazenados. Trata-se, portanto, de uma forma pró-ativa de implementação, não postergando uma importante necessidade para o momento futuro. O processo de expurgo também terá um formato genérico, ou seja, a única necessidade de especificação é o período de retenção dos dados do objeto gerenciado. O modelo interpretará esta especificação e realizará o expurgo de todos os dados de acordo com o critério definido.

A implementação da função de expurgo é feita através de uma *stored procedure*, sendo que as execuções desta função também são atendidas pela "Programação Periódica" (característica mencionada anteriormente). Conforme comentado, a *stored procedure* de expurgo também é genérica (atende todos os dados históricos e alertas armazenados). Esta *stored procedure* não precisa ser adequada para casos específicos de dados coletados pois utiliza um único critério que é o período de retenção. Mudanças nestes critérios (períodos de retenção) são automaticamente assimiladas pelo modelo, não havendo necessidade de nenhum ajuste na *stored procedure*. Estas características não impedem, entretanto, que períodos de retenção superestimados sejam definidos. Ainda assim, ao se constatar o equívoco inicial, bastará o ajuste destes períodos para que os dados desnecessários sejam imediatamente eliminados da base de dados.

A cada execução desta *stored procedure* somente os registros que ultrapassaram o prazo de retenção é que serão eliminados. Desta forma, a periodicidade de execução desta *stored procedure* poderá ser definida sem restrições. Sugere-se, contudo, que não se utilizem intervalos muito longos pois o volume de registros a serem eliminados poderá ser elevado. Considera-se que a execução diária (ou semanal) seja a mais adequada, pois freqüentemente haverão registros que ultrapassaram o período de retenção.

A conversão dos períodos de retenção será feita para dias (menor período). A relação entre o tipo do período para a sumarização e para o expurgo deve ser considerada. Se o critério de sumarização for mensal, não faria sentido utilizar uma retenção baseada em dias (pois somente haverão registros mensais armazenados). Recomenda-se utilizar para este critério o mesmo período ou um período mais abrangente. A tabela 3.2 resume estas recomendações:

Tabela 3.2 – Períodos recomendados

Sumarização	**Retenção**
H, D	D, S, M, A
S	S, M, A
M	M, A
A	A

LEGENDA
H=hora, D=dia, S=semana, M=mês, A=ano

Na descrição do modelo de dados será explanado como a violação destas recomendações será impedida.

Formato geral da *stored procedure* Expurgo

Declaração
Identificação da *stored procedure*: Expurgo
Variáveis de trabalho e cursor Gerenciamento

Execução
Atualização data/hora início última execução
Para cada Gerenciamento:
- Cálculo do número de dias de retenção
- Eliminação dos registros de alerta que ultrapassaram o período de retenção
- Eliminação dos registros históricos que ultrapassaram o período de retenção
Atualização data/hora término última execução

3.3.6 Notificações

A partir da gravação dos alertas em uma tabela do banco de dados, as notificações poderiam ser realizadas de diversas formas. Dependendo do SGBD e do Sistema Operacional utilizados, existiriam alternativas variadas que desempenhariam esta função. Envio de *e-mail*, caixa de mensagem para a estação de trabalho, *trap* para a console de gerenciamento (já existente) seriam algumas destas alternativas.

Este modelo propõe a utilização de estações de gerenciamento para o recebimento de notificações do módulo implementado no SGBD. Uma forma otimizada de gerenciamento é implementada através da definição de grupos de instâncias de bancos de dados. Cada estação de gerenciamento mantém os seus próprios grupos (com as instâncias desejadas). Esta característica torna bastante flexível a definição de atribuições para os gerentes humanos (facilitando a distribuição das tarefas de administração dos bancos de dados). Cada estação de gerenciamento será dotada de um *script* (escrito na linguagem nativa do ambiente) encarregado da verificação e envio das notificações (quando necessário). Para a consulta à base de dados serão utilizadas as ferramentas disponibilizadas por cada SGBD.

A primeira categoria de notificação trata dos alertas propriamente ditos, ou seja, cada estação de gerenciamento estará recebendo uma mensagem indicando a existência de novos alertas para determinada instância gerenciada. A iniciativa de verificação destes alertas é da própria estação de gerenciamento, ou seja, haverá um processo *background*

executando estas verificações periodicamente e enviando uma notificação visual para o gerente humano (quando da ocorrência de novos alertas).

A segunda categoria de notificação é voltada para a verificação da disponibilidade do modelo de gerenciamento implementado. Modelos baseados unicamente na recepção de erros para a identificação de problemas trazem uma desvantagem singular. Se o próprio módulo de gerenciamento apresentar problemas ou se não for possível receber a mensagem de erro (por qualquer motivo) poderá ser ocultada uma situação crítica. Este modelo implementa, então, esta categoria de notificação que resume a situação do módulo de gerenciamento para cada instância.

Esta notificação não será enviada apenas quando da detecção de erros, mas sim em intervalos pré-definidos pelo gerente humano. Cada instância gerenciada será verificada e apresentada com um *status* específico (instância inativa, gerenciamento pendente, regra de gerenciamento com erro e gerenciamento ativo sem erros). As três primeiras situações indicam problemas em potencial com o ambiente gerenciado e deverão ser verificadas e corrigidas.

O recebimento do *status* que comprova que o gerenciamento está ativo e executando todas as regras sem erros é a premissa que o gerente humano deverá adotar. O não recebimento desta notificação será o indício de que alguma anomalia no ambiente está impedindo o correto funcionamento do módulo de gerenciamento (desencadeando uma ação de verificação pelo gerente humano).

Formato geral do *script* Monitoração

Declaração
Cabeçalho de Identificação
Apresentação da *sintaxe* de utilização

Manutenção dos grupos de instâncias
Visualização:
- Das instâncias de determinado grupo
- Dos grupos existentes
Inclusão:
- De uma instância para um determinado grupo
- De um grupo
Remoção:
- De uma instância de um determinado grupo
- De um grupo e de todas as suas instâncias

Monitoração
Teste de conexão com as instâncias do grupo (identificação das inativas)
Chamada da *stored procedure* de Verificação
Identificação de novos alertas para determinada(s) instância(s)
- Notificação visual ao gerente humano (novos alertas)
Resumo do *status* de gerenciamento das instâncias do grupo
Fechamento do ciclo (período de *refresh*)
- Notificação visual ao gerente humano (resumo *status* gerenciamento)

Formato geral da *stored procedure* Verificação

Declaração
Identificação da *stored procedure*: Verificacao
Parâmetros:
- *Refresh* (h) do *status* de gerenciamento das instâncias do grupo
- Instância gerenciada
- Monitor (estação de gerenciamento)
Variáveis de trabalho e cursor Novo_alerta

Execução
Eliminação de eventuais monitores desconectados há mais de 72 horas
Criação do monitor (estação de gerenciamento) – se necessário
Seleção das datas de última execução (Alertas e Gerenciamento)
Seleção e listagem dos novos alertas
Atualização data/hora última execução (Alertas)
Contagem das regras (com erros)
Emissão do resultado da contagem das regras
Ao atingir o período de *refresh* especificado via parâmetro
- Atualização data/hora última execução (Gerenciamento)
- Ativação de *flag* de envio

3.3.7 Ações Corretivas

De um modo geral, este modelo se dispõe a alertar o gerente humano quando da detecção de algum problema. Permite também a iniciativa pró-ativa, ou seja, mesmo não tendo ocorrido nenhum problema poderá antecipar uma provável ocorrência de erro (baseando-se no comportamento de determinado objeto gerenciado).

Comandos DML (*Data Manipulation Language*) podem ser utilizados sem restrições nas *stored procedures* para corrigir/contornar alguns casos. Outros tipos de comandos (*Data Definition Language* ou *System Control Statement*, por exemplo) que também podem ser utilizados para remediar determinadas situações encontradas, poderão

depender do SGBD utilizado. Existem determinadas *features* que permitirão a implementação destas ações corretivas a partir da própria *stored procedure* que identificou o problema.

Neste caso, a correção/adaptação da situação seria realizada pela regra de emissão de alerta. Mesmo tendo uma ação corretiva automática desencadeada a emissão do alerta deverá ser mantida, para que o administrador tome conhecimento da freqüência com que determinado problema vem ocorrendo.

.

3.4 Modelo de dados

As entidades básicas requeridas pelo modelo são:

3.4.1 Gerenciamento

Cadastro de regras programadas para cada tipo de gerenciamento.

Descrição	Formato	Tamanho	Observação
Identificador do Gerenciamento	Numérico	4	Chave Primária
Indicador do tipo de Gerenciamento	Caracter	1	H, A, E, S
Descrição do Gerenciamento	Caracter	50	
Início da última execução	Data/hora	N.A.	
Fim da última execução	Data/hora	N.A.	
Indicador do período de sumarização	Caracter	1	H, D, S, M, A
Indicador do tipo de sumarização	Caracter	1	U, I, A
Indicador do período de retenção	Caracter	1	D, S, M, A
Quantidade de períodos de retenção	Numérico	4	> 0

A chave primária é composta por um único campo (Identificador do Gerenciamento). Os campos com formato Data/hora ocuparão tamanhos diferenciados (dependendo de cada SGBD). Todos os campos possuem *constraints* de verificação que impedem a inclusão de valores nulos (o campo quantidade de períodos de retenção requer um valor maior que zero). Os campos Indicadores deverão conter, obrigatoriamente, uma das entradas válidas conforme abaixo:

- Indicador do tipo de gerenciamento:
  H (histórico), A (alerta), E (expurgo), S (sumarização).

- Indicador do período de sumarização:
  H (hora), D (dia), S (semana), M (mês), A (ano).

- Indicador do tipo de sumarização:
  U (último), I (mínimo), A (máximo).

- Indicador do período de retenção:
  D (dia), S (semana), M (mês), A (ano).

A grande maioria dos tipos de gerenciamento serão ‘H’ (coleta de dados históricos) e ‘A’ (emissão de alertas). Os tipos ‘E’ e ‘S’ definem duas funções especiais do modelo proposto (expurgo e sumarização, respectivamente). Estes dois gerenciamentos já são fornecidos pelo modelo, diferentemente das regras customizadas que são programadas após a implementação. Existirão duas entradas reservadas nesta estrutura para estas duas funções (identificadores 998 e 999).

Para garantir a utilização de período de retenção igual ou de maior abrangência que o de sumarização existe uma *trigger* de verificação específica. O Indicador do período de retenção não permite o valor ‘H’ (existente para sumarização). Este período (hora) não representa um valor razoável para uma funcionalidade que prevê a utilização de períodos mais extensos para a obtenção de resultados mais eficientes. Para melhor desempenho da função de sumarização existe um índice auxiliar para os campos Indicador do tipo de Gerenciamento e Indicador do período de sumarização.

3.4.2 Histórico

Estrutura genérica para dados históricos dos objetos gerenciados.

Descrição	**Formato**	**Tamanho**	**Observação**
Identificador do Gerenciamento	Numérico	4	Chave Estrangeira
Data do Histórico	Data/hora	N.A.	
Descrição complementar	Caracter	100	
Valor do registro	Caracter	200	Formato convertido
Indicador de dado sumarizado	Caracter	1	S, N

A chave primária é composta pelos campos Identificador do Gerenciamento, Data do Histórico e Descrição complementar. O Identificador do Gerenciamento possui o mesmo formato e tamanho do campo homônimo da tabela Gerenciamento (chave estrangeira). A Descrição Complementar somente será utilizada para adequar a composição dos nomes de objetos gerenciados (múltiplas entradas para o mesmo Gerenciamento). Todos os campos possuem *constraints* de verificação que impedem a inclusão de valores nulos (exceto a Descrição Complementar que é opcional). O Indicador de dado sumarizado deverá conter apenas uma das entradas válidas listadas acima: S (sim), N (não). Para melhor desempenho da função de sumarização existe um índice auxiliar para o Indicador de dado sumarizado.

3.4.3 Alerta

Mensagem de aviso ou erro identificado na coleta realizada pela *stored procedure* de gerenciamento.

Descrição	Formato	Tamanho	Observação
Identificador do Gerenciamento	Numérico	4	Chave Estrangeira
Data do Alerta	Data/hora	N.A.	
Descrição complementar	Caracter	100	
Valor do registro	Caracter	200	Formato convertido
Descrição do Alerta	Caracter	300	
Indicador de dado sumarizado	Caracter	1	S, N

A chave primária é composta pelos campos Identificador do Gerenciamento, Data do Alerta e Descrição complementar. O Identificador do Gerenciamento possui o mesmo formato e tamanho do campo homônimo da tabela Gerenciamento (chave estrangeira). Todos os campos possuem *constraints* de verificação que impedem a inclusão de valores nulos. A Descrição Complementar é utilizada para os casos de múltiplas entradas para o mesmo objeto gerenciado. O Indicador de dado sumarizado deverá conter apenas uma das entradas válidas listadas acima: S (sim), N (não). Para melhor desempenho da função de sumarização existe um índice auxiliar para o Indicador de dado sumarizado.

3.4.4 Monitor

Cadastro dinâmico das estações de gerenciamento (monitores).

Descrição	Formato	Tamanho	Observação
Identificação do monitor	Caracter	50	Chave Primária
Última execução alerta	Data/hora	N.A.	
Última execução gerenciamento	Data/hora	N.A.	

A chave primária é composta por um único campo (Identificação do monitor). Os campos com formato Data/hora ocuparão tamanhos diferenciados (dependendo de cada SGBD). Todos os campos possuem *constraints* de verificação que impedem a inclusão de valores. A *stored procedure* de Verificação se encarrega de eliminar os registros de monitores desconectados há mais de 72 horas. Os campos utilizados nesta pesquisa são Última execução alerta e Última execução gerenciamento, mas a quantidade reduzida de registros desta entidade não justifica a criação de um índice auxiliar exclusivo para este fim.

3.5 Vantagens

O modelo proposto apresenta uma alternativa diferenciada para o gerenciamento de bases de dados SQL. Dentre as características do modelo, por se tratar de uma definição "caseira", destaca-se a independência de fornecedor. Este módulo de gerenciamento foi desenvolvido na linguagem SQL (através de *stored procedures*). Não existe nenhum produto ou ferramenta que precise ser adquirida do fornecedor, pois toda estrutura necessária já é integrada ao próprio SGBD (ou ao Sistema Operacional, em alguns casos).

A linguagem SQL é interpretada em tempo de execução (não existem programas compilados). Esta característica poderia trazer uma desvantagem do ponto de vista comercial (dificuldade em criar um produto "fechado", pois seria necessário agregar uma camada exclusiva a este fim). A vantagem em disponibilizar um código-fonte aberto (similar a outros produtos existentes no mercado) tem seu ponto mais forte na facilidade de cooperação de outros especialistas para o produto desenvolvido. Exemplo: a implementação de uma regra de gerenciamento pode ser realizada a partir de um *script* fornecido por algum DBA ou localizado na Internet ou outras fontes de informação (manuais, livros, etc.).

O módulo de gerenciamento proposto não reside numa solução instalada no sistema operacional. Esta característica não onera o modelo com necessidades específicas de plataformas distintas, pois o módulo é incorporado ao SGBD. A capacidade de *schedule* do sistema operacional somente será utilizada quando não for implementada internamente pelo SGBD. Todas as demais definições do módulo de gerenciamento (implementadas no servidor) são internas, ou seja, residentes no próprio SGBD.

Como o módulo de gerenciamento reside no próprio SGBD existe uma grande facilidade na adequação do modelo para plataformas diferentes (portabilidade). A similaridade da linguagem SQL também facilita a implementação do modelo nos diversos bancos de dados existentes (fornecedores diferentes).

A composição das regras é a maior vantagem desta alternativa. Não existem limitações de espécie alguma para esta composição. A programação através da linguagem SQL permite uma customização sem restrições. Necessidades específicas são rapidamente atendidas através da elaboração da *stored procedure* e programação das execuções periódicas.

3.6 Resumo

Neste capítulo foi apresentada uma proposta de modelo de gerenciamento para bases de dados SQL. Foi explanada sua adequação para o modelo genérico de gerenciamento de redes, através do mapeamento de seus componentes. Foram destacadas suas características e vantagens, sendo que, no próximo capítulo, será descrita a implementação deste modelo

proposto, ou seja, demonstrar a utilização deste modelo em um ambiente de bancos de dados existente.

4 Implementação do Modelo Proposto

Este capítulo visa demonstrar a viabilidade de implementação do modelo de gerenciamento proposto descrevendo o ambiente utilizado e os passos para a instalação e customização do módulo de gerenciamento.

4.1 Ambiente de Desenvolvimento

O ambiente utilizado foi o parque de servidores da empresa "X" (por razões de privacidade esta empresa não pode ser identificada). Foi realizada uma seleção entre os servidores disponíveis visando compor um ambiente não apenas quantitativo mas, principalmente, qualitativo. O objetivo foi compor um conjunto de servidores que reunissem as diferentes situações e características a serem validadas no modelo proposto.

O SGBD predominante neste ambiente é o Oracle (com *releases* distintas, mas todas compreendidas na versão 8 deste SGBD). Uma das características do SGBD Oracle é sua total transparência, independente do sistema operacional utilizado. Seja utilizando Unix, Linux ou Windows sempre é mantida a mesma estrutura interna e concepção. Informações podem variar quando decorrentes do sistema operacional utilizado, mas todas entidades Oracle apresentam o mesmo comportamento (independente da plataforma).

Esta característica traz como ponto positivo a facilidade de definição de um ambiente de desenvolvimento. Ou seja, é possível utilizar servidores Unix/Linux/Windows ou até mesmo uma estação de trabalho (através da versão *Personal*). A validação do módulo de gerenciamento é perfeitamente factível em qualquer um destes ambientes citados.

Ainda assim, existem algumas poucas características no modelo proposto que poderão requerer funcionalidades do sistema operacional. Desta forma, a utilização de mais de um ambiente operacional trará maiores benefícios à validação deste modelo. Assim sendo, os ambientes destinados a este fim serão da plataforma Unix e Windows, justamente por apresentarem diferenças significativas entre si (proporcionando maiores condições para submissão a testes do modelo proposto).

Na plataforma Intel são utilizados servidores Compaq (Proliant) com sistema operacional Windows (NT 4.0). Na plataforma Unix são utilizados servidores Sun Enterprise (E-450 e E-4000) com sistema operacional Sun-Solaris (SunOS 5.6 e 5.7). A tabela 4.1 resume a configuração do ambiente de servidores e banco de dados existentes atualmente:

Tabela 4.1 – Ambiente de Desenvolvimento (Servidores)

#	Modelo Servidor	Plataforma/S.O.	Oracle	Instância	Descrição
01	Compaq Proliant 6400R	Intel/Windows NT 4.0	8.1.7	CPQD01	Desenvolvimento-SAP R/3
02			8.1.7	CPQD02	Desenvolvimento-SAP APO
03	Compaq Proliant 6400R	Intel/Windows NT 4.0	8.0.5	CPQT01	Testes-SAP R/3
04	Compaq Proliant DL580	Intel/Windows NT 4.0	8.1.7	CPQD03	Desenvolvimento-SAP EBP
05			8.1.7	CPQD04	Desenvolvimento-SAP WKP
06			8.1.7	CPQD05	Desenvolvimento-SAP BPS
07			8.1.7	CPQD06	Desenvolvimento-SAP catálogo
08	Compaq Proliant DL580	Intel/Windows NT 4.0	8.1.7	CPQT03	Testes-SAP EBP
09			8.1.7	CPQT04	Testes-SAP WKP
10			8.1.7	CPQT05	Testes-SAP BPS
11			8.1.7	CPQT06	Testes-SAP catálogo
12	Sun Enterprise (E-450)	Unix/SunOS 5.7	8.1.7	SUND01	Desenvolvimento-Adm/Fabril
13	Sun Enterprise (E-4000)	Unix/SunOS 5.6	8.1.7	SUND03	Desenvolvimento-SAP RH
14	Compaq Proliant 8500	Intel/Windows NT 4.0	8.0.5	CPQP01	Produção-Cluster-SAP R/3 (A)
15	Compaq Proliant 8500	Intel/Windows NT 4.0	8.0.5	CPQP01	Produção-Cluster-SAP R/3 (B)
16	Compaq Proliant 8500	Intel/Windows NT 4.0	8.0.5	CPQP02	Produção-SAP APO
17	Compaq Proliant DL760	Intel/Windows NT 4.0	8.1.7	CPQP03	Produção-SAP EBP
18			8.1.7	CPQP05	Produção-SAP BPS
19	Compaq Proliant DL380	Intel/Windows NT 4.0	8.1.7	CPQP04	Produção-SAP WKP
20			8.1.7	CPQP06	Produção-SAP catálogo
21	Sun Enterprise (E-450)	Unix/SunOS 5.6	8.1.7	SUNP01	Produção-Administrativo
22	Sun Enterprise (E-450)	Unix/SunOS 5.7	8.1.7	SUNP02	Produção-Fabril
23			8.1.7	SUNB01	Produção-Catálogo Backup
24	Sun Enterprise (E-450)	Unix/SunOS 5.6	8.1.7	SUNP03	Produção-SAP RH

Efetuando-se um cruzamento entre as informações da tabela apresentada no item anterior (Plataforma/S.O. X Oracle) obtém-se as seguintes combinações possíveis:

- Intel/Windows NT 4.0 Oracle 8.0.5
- Intel/Windows NT 4.0 Oracle 8.1.7
- Unix/SunOS 5.6 Oracle 8.1.7
- Unix/SunOS 5.7 Oracle 8.1.7

A estratégia adotada considerou as combinações acima e priorizou o contexto qualitativo. Concentrou-se, desta forma, a implementação em quatro instâncias produtivas que compõem um conjunto representativo para a validação do modelo. Ao invés de concentrar o esforço na implementação do módulo de gerenciamento em todas as instâncias, foi realizada a implementação neste conjunto de instâncias e priorizada a coleta e análise dos resultados obtidos com o modelo implementado.

Antes, porém, da implementação deste modelo nas instâncias produtivas, foi desenvolvido um projeto piloto em uma ambiente de testes. Para esta finalidade foi utilizado o ambiente #12 constante na tabela acima.

Após a depuração do modelo e exaustiva submissão de testes no ambiente piloto iniciou-se a implementação nos ambientes produtivos. Conforme o critério descrito acima, foram selecionadas (inicialmente) quatro instâncias consideradas significativas para a validação do modelo proposto. Estas instâncias estão identificadas na tabela de ambientes com os números 14/15, 17, 21 e 22. A primeira instância produtiva (identificada pelos números 14/15) está associada a dois servidores por se tratar de um ambiente clusterizado (composto por dois nós).

4.2 Instalação do Modelo

A etapa inicial de implementação do modelo requer a criação de um usuário no banco de dados para que seja o proprietário das entidades que compõem o módulo de gerenciamento. Este usuário deverá ser dotado de privilégios que permitam a consulta nas tabelas e visões do dicionário de dados do SGBD, além de execução de comandos que permitam ações corretivas. A sintaxe abaixo descreve como foi feita a criação deste usuário nos ambientes que tiveram o modelo implementado.

Criação do usuário proprietário

```
CREATE USER GBD IDENTIFIED BY GERSQL01
   DEFAULT TABLESPACE TOOLS TEMPORARY TABLESPACE TEMP;
GRANT CONNECT, RESOURCE, DBA TO GBD;
```

Na seqüência foram criadas as tabelas descritas no capítulo anterior (ver Modelo de Dados). Estas tabelas são as entidades básicas requeridas pelo modelo e foram definidas conforme *script* abaixo:

Criação das entidades do repositório de dados

Gerenciamento

```
CREATE TABLE GERENCIAMENTO (
ID_GRC          NUMBER(4)       NOT NULL,
IC_TIP_GRC      VARCHAR2(1)     NOT NULL,
DE_GRC          VARCHAR2(50)    NOT NULL,
DT_UEX_INI      DATE            NOT NULL,
DT_UEX_FIM      DATE            NOT NULL,
IC_PER_SMZ      VARCHAR2(1)     NOT NULL,
IC_TIP_SMZ      VARCHAR2(1)     NOT NULL,
IC_PER_RTC      VARCHAR2(1)     NOT NULL,
QT_PER_RTC      NUMBER(4)       NOT NULL
);
alter table GERENCIAMENTO add CONSTRAINT GER_PK
   PRIMARY KEY (ID_GRC);
alter table GERENCIAMENTO add CONSTRAINT GER_CH_IC_TIP_GRC
   CHECK (IC_TIP_GRC IN ('H','A','E','S'));
alter table GERENCIAMENTO add CONSTRAINT GER_CH_IC_PER_SMZ
   CHECK (IC_PER_SMZ IN ('H','D','S','M','A'));
alter table GERENCIAMENTO add CONSTRAINT GER_CH_IC_TIP_SMZ
   CHECK (IC_TIP_SMZ IN ('U','I','A'));
alter table GERENCIAMENTO add CONSTRAINT GER_CH_IC_PER_RTC
   CHECK (IC_PER_RTC IN ('D','S','M','A'));
alter table GERENCIAMENTO add CONSTRAINT GER_CH_QT_PER_RTC
   CHECK (QT_PER_RTC > 0);
create index GER_IX_IC_TIP_GRC on GERENCIAMENTO (IC_TIP_GRC, IC_PER_SMZ);
```

```
CREATE TRIGGER TB_GERENCIAMENTO
   BEFORE INSERT OR UPDATE
   ON GERENCIAMENTO FOR EACH ROW
DECLARE
   AX_MAX NUMBER;
BEGIN
   IF (:NEW.IC_PER_SMZ='H' OR :NEW.IC_PER_SMZ='D')
      AND :NEW.IC_PER_RTC NOT IN ('D','S','M','A') THEN
      RAISE_APPLICATION_ERROR
         (-20999,'PARA IC_PER_SMZ=H/D IC_PER_RTC DEVE SER=D/S/M/A');
   ELSE
      IF :NEW.IC_PER_SMZ='S' AND :NEW.IC_PER_RTC NOT IN ('S','M','A') THEN
         RAISE_APPLICATION_ERROR
            (-20999,'PARA IC_PER_SMZ=S IC_PER_RTC DEVE SER=S/M/A');
      ELSE
         IF :NEW.IC_PER_SMZ='M' AND :NEW.IC_PER_RTC NOT IN ('M','A') THEN
            RAISE_APPLICATION_ERROR
               (-20999,'PARA IC_PER_SMZ=M IC_PER_RTC DEVE SER=M/A');
         ELSE
            IF :NEW.IC_PER_SMZ='A' AND :NEW.IC_PER_RTC <> 'A' THEN
               RAISE_APPLICATION_ERROR
                  (-20999,'PARA IC_PER_SMZ=A IC_PER_RTC DEVE SER=A');
            END IF;
         END IF;
      END IF;
   END IF;
END;
```

Histórico

```
CREATE TABLE HISTORICO (
ID_GRC          NUMBER(4)       NOT NULL,
DT_HST          DATE            NOT NULL,
DE_CMP          VARCHAR2(100),
VL_REG          VARCHAR2(200)   NOT NULL,
IC_DAD_SMZ      VARCHAR2(1)     NOT NULL
);
alter table HISTORICO add CONSTRAINT HST_PK
   PRIMARY KEY (ID_GRC, DT_HST, DE_CMP);
alter table HISTORICO add CONSTRAINT HST_FK_ID_GRC
   FOREIGN KEY (ID_GRC) REFERENCES GERENCIAMENTO (ID_GRC);
alter table HISTORICO add CONSTRAINT HST_CH_IC_DAD_SMZ
   CHECK (IC_DAD_SMZ IN ('S','N'));
create index HST_IX_IC_DAD_SMZ on HISTORICO (IC_DAD_SMZ);

CREATE VIEW V_HISTORICO AS
SELECT H.ID_GRC, H.DT_HST, G.DE_GRC, H.DE_CMP, H.VL_REG, H.IC_DAD_SMZ
   FROM HISTORICO H, GERENCIAMENTO G
   WHERE H.ID_GRC=G.ID_GRC;
```

Alerta

```
CREATE TABLE ALERTA (
ID_GRC          NUMBER(4)       NOT NULL,
DT_ALE          DATE            NOT NULL,
DE_CMP          VARCHAR2(100),
VL_REG          VARCHAR2(200)   NOT NULL,
DE_ALE          VARCHAR2(300)   NOT NULL,
IC_DAD_SMZ      VARCHAR2(1)     NOT NULL
);
alter table ALERTA add CONSTRAINT ALE_PK
   PRIMARY KEY (ID_GRC, DT_ALE, DE_CMP);
alter table ALERTA add CONSTRAINT ALE_FK_ID_GRC
   FOREIGN KEY (ID_GRC) REFERENCES GERENCIAMENTO (ID_GRC);
alter table ALERTA add CONSTRAINT ALE_CH_IC_DAD_SMZ
   CHECK (IC_DAD_SMZ IN ('S','N'));
create index ALE_IX_IC_DAD_SMZ on ALERTA (IC_DAD_SMZ);

CREATE VIEW V_ALERTA AS
SELECT A.ID_GRC, A.DT_ALE, G.DE_GRC, A.DE_CMP, A.VL_REG, A.DE_ALE,  A.IC_DAD_SMZ
   FROM ALERTA A, GERENCIAMENTO G
   WHERE A.ID_GRC=G.ID_GRC;
```

Monitor

```
CREATE TABLE MONITOR (
DE_MON           VARCHAR2(50)    NOT NULL,
DT_UEX_ALE       DATE            NOT NULL,
DT_UEX_GRC       DATE            NOT NULL
);
alter table MONITOR add CONSTRAINT MON_PK
   PRIMARY KEY (DE_MON);
```

Conforme mencionado no capítulo anterior, a tabela Gerenciamento possui uma *trigger* associada que impede a utilização de um período de sumarização de menor abrangência que o período de retenção. Para as tabelas Histórico e Alerta foram criadas visões para incorporar a descrição do Gerenciamento.

As funções de sumarização e expurgo foram criadas no passo seguinte. Além da codificação destas *stored procedures* (que integram o modelo) fez-se necessário também, o cadastramento na entidade Gerenciamento, bem como a definição destas execuções através da "Programação Periódica".

Criação da *stored procedure* de Sumarização

```
-- ********************************************************************************
--
--                          SUMARIZAÇÃO
--
-- ********************************************************************************
```

Identificação da *stored procedure*: Sumarizacao

```
CREATE OR REPLACE PROCEDURE SUMARIZAÇÃO
```

Parâmetro de entrada (Indicador do período de sumarização)

```
   (AX_VL_PRM_001 IN VARCHAR2) IS
```

Variáveis de trabalho e cursores Histórico e Alerta

```
   AX_ID_GRC         NUMBER:=998;
   AX_DA_REF         VARCHAR2(12);
   AX_VL_REF         VARCHAR2(12);
   AX_ID_GRC_ANT     NUMBER(3):=0;
   AX_DT_HST_ANT     DATE;
   AX_DE_CMP_ANT     VARCHAR2(100);
   AX_VL_REG_ANT     VARCHAR2(200);
   AX_DT_ALE_ANT     DATE;
   AX_DE_ALE_ANT     VARCHAR2(300);
   CURSOR HISTORICO IS
      SELECT H.ID_GRC, H.DE_CMP, H.DT_HST, H.VL_REG, G.IC_TIP_SMZ
         FROM HISTORICO H, GERENCIAMENTO G
            WHERE G.ID_GRC=H.ID_GRC AND G.IC_TIP_GRC='H'
            AND G.IC_PER_SMZ=AX_VL_PRM_001 AND H.IC_DAD_SMZ='N'
               ORDER BY 1, 2, 3, 4;
   CURSOR ALERTA IS
      SELECT A.ID_GRC, A.DE_CMP, A.DT_ALE, A.VL_REG, A.DE_ALE, G.IC_TIP_SMZ
         FROM ALERTA A, GERENCIAMENTO G
            WHERE G.ID_GRC=A.ID_GRC AND G.IC_TIP_GRC='A'
            AND G.IC_PER_SMZ=AX_VL_PRM_001 AND A.IC_DAD_SMZ='N'
               ORDER BY 1, 2, 3, 4;
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Sumarização Historico
```

Formatação da data de referência para sumarização conforme parâmetro

```
   IF AX_VL_PRM_001='A' THEN
      AX_DA_REF:='YYYY';
```

```
   ELSE
      IF AX_VL_PRM_001='M' THEN
         AX_DA_REF:='YYYYMM';
      ELSE
         IF AX_VL_PRM_001='S' THEN
            AX_DA_REF:='YYYYWW';
         ELSE
            IF AX_VL_PRM_001='D' THEN
               AX_DA_REF:='YYYYMMDD';
            ELSE
               IF AX_VL_PRM_001='H' THEN
                  AX_DA_REF:='YYYYMMDDHH24';
               END IF;
            END IF;
         END IF;
      END IF;
   END IF;
   AX_VL_REF:=TO_CHAR(SYSDATE,AX_DA_REF);
```

Seleção dos registros históricos (preservando o mais significativo)

```
   FOR HST IN HISTORICO LOOP
      IF TO_CHAR(HST.DT_HST,AX_DA_REF) <> AX_VL_REF THEN
         IF AX_ID_GRC_ANT <> 0
            AND ((AX_ID_GRC_ANT <> HST.ID_GRC)
               OR (TO_CHAR(AX_DT_HST_ANT,AX_DA_REF) <> TO_CHAR(HST.DT_HST,AX_DA_REF))
               OR (AX_DE_CMP_ANT <> HST.DE_CMP)) THEN
            INSERT INTO HISTORICO VALUES
               (AX_ID_GRC_ANT, AX_DT_HST_ANT, AX_DE_CMP_ANT, AX_VL_REG_ANT, 'S');
```

Eliminação dos demais registros históricos correspondentes ao período sumarizado

```
            DELETE FROM HISTORICO
               WHERE ID_GRC=AX_ID_GRC_ANT AND NVL(DE_CMP,0)=NVL(AX_DE_CMP_ANT,0)
               AND TO_CHAR(DT_HST,AX_DA_REF)=TO_CHAR(AX_DT_HST_ANT,AX_DA_REF)
               AND IC_DAD_SMZ='N';
            AX_ID_GRC_ANT:=0;
            COMMIT;
         END IF;
         IF AX_ID_GRC_ANT = 0
            OR ((HST.IC_TIP_SMZ='U')
               OR (HST.IC_TIP_SMZ='I' AND HST.VL_REG <= AX_VL_REG_ANT)
               OR (HST.IC_TIP_SMZ='A' AND HST.VL_REG >= AX_VL_REG_ANT)) THEN
            AX_DT_HST_ANT:=HST.DT_HST;
            AX_VL_REG_ANT:=HST.VL_REG;
         END IF;
         AX_ID_GRC_ANT:=HST.ID_GRC;
         AX_DE_CMP_ANT:=HST.DE_CMP;
      END IF;
   END LOOP;
   IF AX_ID_GRC_ANT <> 0 THEN
      INSERT INTO HISTORICO VALUES
         (AX_ID_GRC_ANT, AX_DT_HST_ANT, AX_DE_CMP_ANT, AX_VL_REG_ANT, 'S');
      DELETE FROM HISTORICO
         WHERE ID_GRC=AX_ID_GRC_ANT AND NVL(DE_CMP,0)=NVL(AX_DE_CMP_ANT,0)
         AND TO_CHAR(DT_HST,AX_DA_REF)=TO_CHAR(AX_DT_HST_ANT,AX_DA_REF)
         AND IC_DAD_SMZ='N';
      COMMIT;
   END IF;
-- Sumarização Alerta
   AX_ID_GRC_ANT:=0;
```

Seleção dos registros de alerta (preservando o mais significativo)

```
   FOR ALE IN ALERTA LOOP
      IF TO_CHAR(ALE.DT_ALE,AX_DA_REF) <> AX_VL_REF THEN
         IF AX_ID_GRC_ANT <> 0
            AND ((AX_ID_GRC_ANT <> ALE.ID_GRC)
               OR (TO_CHAR(AX_DT_ALE_ANT,AX_DA_REF) <> TO_CHAR(ALE.DT_ALE,AX_DA_REF))
               OR (AX_DE_CMP_ANT <> ALE.DE_CMP)) THEN
            INSERT INTO ALERTA VALUES
               (AX_ID_GRC_ANT, AX_DT_ALE_ANT, AX_DE_CMP_ANT, AX_VL_REG_ANT,
               AX_DE_ALE_ANT, 'S');
```

Eliminação dos demais registros de alerta correspondentes ao período sumarizado

```
            DELETE FROM ALERTA
```

```
               WHERE ID_GRC=AX_ID_GRC_ANT AND NVL(DE_CMP,0)=NVL(AX_DE_CMP_ANT,0)
               AND TO_CHAR(DT_ALE,AX_DA_REF)=TO_CHAR(AX_DT_ALE_ANT,AX_DA_REF)
               AND IC_DAD_SMZ='N';
            AX_ID_GRC_ANT:=0;
            COMMIT;
         END IF;
         IF AX_ID_GRC_ANT = 0
            OR ((ALE.IC_TIP_SMZ='U')
               OR (ALE.IC_TIP_SMZ='I' AND ALE.VL_REG <= AX_VL_REG_ANT)
               OR (ALE.IC_TIP_SMZ='A' AND ALE.VL_REG >= AX_VL_REG_ANT)) THEN
            AX_DT_ALE_ANT:=ALE.DT_ALE;
            AX_VL_REG_ANT:=ALE.VL_REG;
            AX_DE_ALE_ANT:=ALE.DE_ALE;
         END IF;
         AX_ID_GRC_ANT:=ALE.ID_GRC;
         AX_DE_CMP_ANT:=ALE.DE_CMP;
      END IF;
   END LOOP;
   IF AX_ID_GRC_ANT <> 0 THEN
      INSERT INTO ALERTA VALUES
         (AX_ID_GRC_ANT, AX_DT_ALE_ANT, AX_DE_CMP_ANT, AX_VL_REG_ANT,
         AX_DE_ALE_ANT, 'S');
      DELETE FROM ALERTA
         WHERE ID_GRC=AX_ID_GRC_ANT AND NVL(DE_CMP,0)=NVL(AX_DE_CMP_ANT,0)
         AND TO_CHAR(DT_ALE,AX_DA_REF)=TO_CHAR(AX_DT_ALE_ANT,AX_DA_REF)
         AND IC_DAD_SMZ='N';
      COMMIT;
   END IF;
-- Final do processamento
```

Atualização data/hora término última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
```

Criação da *stored procedure* de Expurgo

```
-- ******************************************************************************
--
--                            EXPURGO
--
-- ******************************************************************************
```

Identificação da *stored procedure*: Expurgo

```
CREATE OR REPLACE PROCEDURE EXPURGO IS
```

Variáveis de trabalho e cursor Gerenciamento

```
   AX_ID_GRC          NUMBER:=999;
   AX_IC_TIP_GRC      VARCHAR2(1);
   AX_NR_DIAS         NUMBER;
   CURSOR GERENCIAMENTO IS
      SELECT ID_GRC, IC_TIP_GRC, IC_PER_RTC, QT_PER_RTC
         FROM GERENCIAMENTO WHERE IC_TIP_GRC IN ('A','H');
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Expurgo
```

Para cada Gerenciamento:

```
   FOR GER IN GERENCIAMENTO LOOP
```

- Cálculo do número de dias de retenção

```
      IF GER.IC_PER_RTC='D' THEN
         AX_NR_DIAS:=GER.QT_PER_RTC;
      ELSE
         IF GER.IC_PER_RTC='S' THEN
            AX_NR_DIAS:=GER.QT_PER_RTC * 7;
         ELSE
            IF GER.IC_PER_RTC='M' THEN
               AX_NR_DIAS:=GER.QT_PER_RTC * 30;
            ELSE
```

```
               IF GER.IC_PER_RTC='A' THEN
                  AX_NR_DIAS:=GER.QT_PER_RTC * 365;
               END IF;
            END IF;
         END IF;
      END IF;
```

- Eliminação dos registros de alerta que ultrapassaram o período de retenção

```
      IF GER.IC_TIP_GRC='A' THEN
         DELETE FROM ALERTA
            WHERE ID_GRC=GER.ID_GRC AND DT_ALE < (SYSDATE - AX_NR_DIAS);
```

- Eliminação dos registros históricos que ultrapassaram o período de retenção

```
      ELSE
         DELETE FROM HISTORICO
            WHERE ID_GRC=GER.ID_GRC AND DT_HST < (SYSDATE - AX_NR_DIAS);
      END IF;
      COMMIT;
   END LOOP;
-- Final do processamento
```

Atualização data/hora término última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
```

Cadastramento na entidade Gerenciamento

```
INSERT INTO GERENCIAMENTO VALUES
   (998,'S','SUMARIZAÇÃO',SYSDATE,SYSDATE,'A','U','A',90);
INSERT INTO GERENCIAMENTO VALUES
   (999,'E','EXPURGO',SYSDATE,SYSDATE,'A','U','A',90);
```

Programação Periódica das execuções (Sumarização e Expurgo)

```
DECLARE
  jobno number := 0;
begin
  dbms_job.submit(jobno,'SUMARIZACAO (A);',sysdate,'sysdate+365');
  dbms_job.submit(jobno,'SUMARIZACAO (M);',sysdate,'sysdate+30');
  dbms_job.submit(jobno,'SUMARIZACAO (S);',sysdate,'sysdate+7');
  dbms_job.submit(jobno,'SUMARIZACAO (D);',sysdate,'sysdate+1');
  dbms_job.submit(jobno,'SUMARIZACAO (H);',sysdate,'sysdate+1/24');
  dbms_job.submit(jobno,'EXPURGO;',sysdate,'sysdate+7');
  commit;
end;
```

Para permitir programações diferenciadas para as sumarizações (dependendo do período) a *stored procedure* recebe um parâmetro (que será utilizado na definição dos cursores Histórico e Alerta). As especificações acima correspondem às definições realizadas no ambiente piloto (como exemplo). A definição do intervalo para as execuções das funções de sumarização e expurgo, entretanto, pode ser realizada livremente (adotando-se valores adequados para cada situação).

A última *stored procedure* a ser criada será utilizada pelas estações de gerenciamento e será responsável pela emissão das notificações. A sintaxe utilizada está listada a seguir:

Criação da *stored procedure* de Verificação

```
-- ********************************************************************************
--
--                              VERIFICACAO
--
-- ********************************************************************************
```

Identificação da *stored procedure*: Verificacao

```
CREATE OR REPLACE PROCEDURE VERIFICACAO
```

Parâmetros:

- *Refresh* (h) do *status* de gerenciamento das instâncias do grupo
- Instância gerenciada
- Monitor (estação de gerenciamento)

```
   (AX_VL_PR1 IN NUMBER, AX_VL_PR2 IN VARCHAR2, AX_VL_PR3 IN VARCHAR2) IS
```

Variáveis de trabalho e cursor Novo_alerta

```
   AX_DT_UEX_ALE      DATE;
   AX_DT_UEX_GRC      DATE;
   AX_DT_ATU          DATE;
   AX_NR_AUX          NUMBER;
   AX_NR_ERR          NUMBER;
   CURSOR NOVO_ALERTA IS
      SELECT A.ID_GRC, A.DT_ALE, G.DE_GRC, A.DE_CMP, A.VL_REG, A.DE_ALE
         FROM ALERTA A, GERENCIAMENTO G
         WHERE A.ID_GRC=G.ID_GRC AND DT_ALE > AX_DT_UEX_ALE
            ORDER BY 2;
BEGIN
```

Eliminação de eventuais monitores desconectados há mais de 72 horas

```
   DELETE FROM MONITOR WHERE DT_UEX_ALE < SYSDATE-4 OR DT_UEX_GRC < SYSDATE-4;
   SELECT COUNT(*) INTO AX_NR_AUX FROM MONITOR
      WHERE DE_MON=AX_VL_PR3;
```

Criação do monitor (estação de gerenciamento) – se necessário

```
   IF AX_NR_AUX=0 THEN
      INSERT INTO MONITOR VALUES (AX_VL_PR3, SYSDATE-3, SYSDATE-3);
   END IF;
   COMMIT;
```

Seleção das datas de última execução (Alertas e Gerenciamento)

```
   SELECT DT_UEX_ALE, DT_UEX_GRC, SYSDATE INTO AX_DT_UEX_ALE, AX_DT_UEX_GRC, AX_DT_ATU
      FROM MONITOR WHERE DE_MON=AX_VL_PR3;
-- Alertas
   AX_NR_ERR:=0;
```

Seleção e listagem dos novos alertas

```
   FOR ALE IN NOVO_ALERTA LOOP
      DBMS_OUTPUT.PUT_LINE('ORA-200' || LPAD(ALE.ID_GRC,3,'0') || ': ' ||
         TO_CHAR(ALE.DT_ALE,'YYYY/MM/DD HH24:MI:SS') || ' ' || ALE.DE_GRC || ' ' ||
         ALE.DE_CMP || ' ' || LTRIM(ALE.VL_REG,' ') || ' ' || ALE.DE_ALE);
      AX_NR_ERR:=AX_NR_ERR+1;
   END LOOP;
   IF AX_NR_ERR > 0 THEN
      DBMS_OUTPUT.PUT_LINE(' {ALE} ' || AX_VL_PR2 || ' ' || AX_NR_ERR);
   END IF;
```

Atualização data/hora última execução (Alertas)

```
   UPDATE MONITOR SET DT_UEX_ALE=AX_DT_ATU WHERE DE_MON=AX_VL_PR3;
-- Gerenciamento
```

Contagem das regras (com erros)

```
   SELECT COUNT(*) INTO AX_NR_ERR FROM SYS.DBA_JOBS
      WHERE LOG_USER='GBD' AND (BROKEN='Y' OR NEXT_DATE < SYSDATE OR FAILURES > 0);
```

Emissão do resultado da contagem das regras

```
   IF AX_NR_ERR=0 THEN
      DBMS_OUTPUT.PUT_LINE('{GRC}{OK} ' || AX_VL_PR2);
   ELSE
      DBMS_OUTPUT.PUT_LINE('{GRC}{ERRO} ' || AX_VL_PR2 || ' ' || AX_NR_ERR);
   END IF;
```

```
Ao atingir o período de refresh especificado via parâmetro
   IF AX_DT_ATU > AX_DT_UEX_GRC + (AX_VL_PR1/24) THEN
- Atualização data/hora última execução (Gerenciamento)
      UPDATE MONITOR SET DT_UEX_GRC=AX_DT_ATU WHERE DE_MON=AX_VL_PR3;
- Ativação de flag de envio
      DBMS_OUTPUT.PUT_LINE('{ENVIAR}');
   END IF;
   COMMIT;
END;
```

Existe um usuário definido como o proprietário das entidades que compõem o modelo. Nas estações de gerenciamento, entretanto, não deverá ser utilizado este usuário pelo fato do mesmo ser dotado de privilégios de administrador. Para tanto deve-se utilizar um usuário específico que tenha apenas privilégios de leitura nas entidades do módulo de gerenciamento (para o projeto piloto adotou-se o usuário MBD). Para permitir que este e outros usuários (que não o proprietário) acessem as informações coletadas e gravadas pelo módulo são criados sinônimos públicos e liberadas autorizações de acesso à estas entidades.

Criação de sinônimos públicos e autorizações

```
CREATE PUBLIC SYNONYM ALERTA        FOR GBD.ALERTA;
CREATE PUBLIC SYNONYM GERENCIAMENTO FOR GBD.GERENCIAMENTO;
CREATE PUBLIC SYNONYM HISTORICO     FOR GBD.HISTORICO;
CREATE PUBLIC SYNONYM HIST          FOR GBD.HIST;
CREATE PUBLIC SYNONYM MONITOR       FOR GBD.MONITOR;
CREATE PUBLIC SYNONYM VERIFICACAO   FOR GBD.VERIFICACAO;
GRANT  SELECT  ON     ALERTA        TO  PUBLIC;
GRANT  SELECT  ON     GERENCIAMENTO TO  PUBLIC;
GRANT  SELECT  ON     HISTORICO     TO  PUBLIC;
GRANT  SELECT  ON     HIST          TO  PUBLIC;
GRANT  SELECT  ON     MONITOR       TO  PUBLIC;
GRANT  EXECUTE ON     VERIFICACAO   TO  PUBLIC;
```

As etapas descritas até este ponto completam todas as necessidades de implementação no SGBD. Já havia sido destacada como característica deste modelo o fato do módulo de gerenciamento residir no próprio SGBD.

Para finalizar a instalação deste modelo resta apenas a definição da(s) estação(ões) de gerenciamento. Neste quesito é utilizado um *script* na linguagem nativa do ambiente. Para a utilização de uma estação de gerenciamento na plataforma Windows foi utilizado o código descrito abaixo:

Criação do *script* de Monitoração

```
@ECHO OFF
:help
cls
Cabeçalho de Identificação
echo ================================================================================
echo / / / / / / / /  GERENCIAMENTO DE BANCOS DE DADOS SQL - MONITOR  \ \ \ \ \ \ \ \
echo ================================================================================
Apresentação da sintaxe de utilização
echo v             - Visualizar grupos existentes
echo v [gr]        - Visualizar instancias do grupo [gr]
echo --------------------------------------------------------------------------------
```

```
echo a [gr]        - Criar grupo [gr]
echo a [gr] [inst] - Adicionar instancia [inst] ao grupo [gr]
echo ------------------------------------------------------------------------------
echo r [gr] [inst] - Remover instancia [inst] do grupo [gr]
echo r [gr]        - Remover grupo [gr]
echo ------------------------------------------------------------------------------
echo m [gr] [seg] [h] - Verificar instancias do grupo [gr] (refresh=[seg] status=[h])
echo ==============================================================================
if '%1' == ''  goto fim
echo               - Parametro(s) recebido(s): %1 %2 %3 %4 %5 %6 %7 %8 %9
echo ------------------------------------------------------------------------------
if '%1' == 'v' goto visualizar
if '%1' == 'a' goto adicionar
if '%1' == 'r' goto remover
if '%1' == 'm' goto monitor
set msg=Parametro invalido
goto erro
```

Manutenção dos grupos de instâncias

Visualização:

```
:visualizar
if '%2' == '' goto ver_grupos
set msg=Grupo inexistente: %2
if not exist "%2" goto erro
```

- Das instâncias de determinado grupo

```
echo               - Visualizando grupo %2
cd %2
for %%f in (*.*) do echo                 - %%f
cd ..
goto fim
```

- Dos grupos existentes

```
:ver_grupos
echo               - Visualizando grupos
dir | findstr -i "<DIR>"
goto fim
```

Inclusão:

```
:adicionar
set msg=Informar grupo/instancia
if '%2' == '' goto erro
if '%3' == '' goto ad_grupo
set msg=Grupo inexistente: %2
if not exist "%2" goto erro
```

- De uma instância para um determinado grupo

```
echo               - Adicionando instancia %2\%3
cd %2
echo > %3
for %%f in (*.*) do echo                 - %%f
cd ..
goto fim
```

- De um grupo

```
:ad_grupo
echo               - Adicionando grupo %2
mkdir %2
dir | findstr -i "<DIR>"
goto fim
```

Remoção:

```
:remover
set msg=Informar grupo/instancia
if '%2' == '' goto erro
set msg=Grupo inexistente: %2
if not exist "%2" goto erro
if '%3' == '' goto rem_grupo
```

- De uma instância de um determinado grupo

```
echo               - Removendo instancia %2\%3
set msg=Instancia inexistente: %2\%3
if not exist "%2\%3" goto erro
cd %2
```

```
del %3
for %%f in (*.*) do echo                  - %%f
cd ..
goto fim
```

- De um grupo e de todas as suas instâncias

```
:rem_grupo
echo              - Removendo grupo %2
rmdir %2 /S /Q
dir | findstr -i "<DIR>"
goto fim
```

Monitoração

```
:monitor
for /F %%M in ('hostname') do set m=%%M
set msg=Grupo inexistente: %2
if not exist "%2" goto erro
set msg=Informar intervalo refresh
if '%3' == '' goto erro
set msg=Intervalo status invalido (1/2/3/4/6/8/12/24 h)
set hor=erro
for %%v in (1 2 3 4 6 8 12 24) do if %4 == %%v set hor=OK
if not %hor% == OK goto erro
:loop
cd %2
if exist "inat" rmdir inat /S /Q
mkdir inat
if exist "pend" rmdir pend /S /Q
mkdir pend
if exist "exec" rmdir exec /S /Q
mkdir exec
if exist "ativ" rmdir ativ /S /Q
mkdir ativ
```

Teste de conexão com as instâncias do grupo (identificação das inativas)

```
for %%f in (*.*) do echo tnsping %%f ^> inat\%%f              >> exec\connect.bat
call exec\connect.bat
cd inat
for %%f in (*.*) do grep "OK (" %%f                           >  ..\pend\%%f
cd ..\pend
dir                                                           >  pend.txt
for /F "Tokens=4" %%f in ('grep " 0 " pend.txt') do del %%f
for %%f in (*.*) do del ..\inat\%%f
```

Chamada da *stored procedure* de Verificação

```
for %%f in (*.*) do echo set serveroutput on size 1000000     >  ..\exec\%%f.sql
for %%f in (*.*) do echo spool %%f.STATUS                     >> ..\exec\%%f.sql
for %%f in (*.*) do echo execute verificacao (%4,'%%f','%m%') >> ..\exec\%%f.sql
for %%f in (*.*) do echo exit;                                >> ..\exec\%%f.sql
for %%f in (*.*) do echo @sqlplus mbd/mbd@%%f @%%f.sql        >> ..\exec\monitor.bat
cd ..\exec
call monitor.bat
```

Identificação de novos alertas para determinada(s) instância(s)

```
:alerta
for /F "Tokens=1,2,3" %%a in ('grep {ALE} *.STATUS') do
```

- Notificação visual ao gerente humano (novos alertas)

```
   net send %m% Grupo %2 - %%b COM %%c NOVO(S) ALERTA(S)
```

Resumo do *status* de gerenciamento das instâncias do grupo

```
:gerenciamento
for /F "Tokens=1,2,3" %%a in ('grep {GRC} *.STATUS') do move ..\pend\%%b ..\ativ\%%b
cd ..\inat
set msg=:nenhum
for %%f in (*.*)                    do set msg=:
if '%msg%' == ':' for %%f in (*.*) do call :msg %%f
set inat=%msg%
cd ..\pend
set msg=:nenhum
for %%f in (*.*)                    do set msg=:
if '%msg%' == ':' for %%f in (*.*) do call :msg %%f
set pend=%msg%
cd ..\exec
```

```
set msg=:nenhum
for /F "Tokens=1,2,3" %%a in ('grep {GRC}{ERRO} *.STATUS') do set msg=:
if '%msg%' == ':' for /F "Tokens=1,2,3" %%a in ('grep {GRC}{ERRO} *.STATUS') do
   call :msg %%b
set erro=%msg%
set msg=:nenhum
for /F "Tokens=1,2,3" %%a in ('grep {GRC}{OK} *.STATUS')   do set msg=:
if '%msg%' == ':' for /F "Tokens=1,2,3" %%a in ('grep {GRC}{OK} *.STATUS') do
   call :msg %%b
set ativ=%msg%
grep ORA- *.STATUS
set msg=Resumo Gerenciamento Grupo %2: * * * INATIVO(S)%inat% * * * PENDENTE(S)%pend%
   * * * ATIVO(S) C/ERRO(S)%erro% * * * ATIVO(S)%ativ%
echo %msg%
set flag=nao
for /F "Tokens=1,2,3" %%a in ('grep {ENVIAR} *.STATUS') do set flag=sim
```

Fechamento do ciclo (período de *refresh*)

- Notificação visual ao gerente humano (resumo *status* gerenciamento)

```
if '%flag%' == 'sim' net send %m% %msg%
cd ..
rmdir inat /S /Q
rmdir pend /S /Q
rmdir exec /S /Q
rmdir ativ /S /Q
cd ..
for /F "Tokens=2" %%D in ('Date /T') do (set DATE=%%D)
for /F "Tokens=4" %%T in ('now')     do (set TIME=%%T)
echo Grupo %2 %DATE% %TIME% (refresh=%3s status=%4h). CTRL-C para encerrar ...
echo ------------------------------------------------------------------------------
set msg=Intervalo refresh invalido
sleep %3
if errorlevel == 1 goto erro
goto loop
:erro
echo                 * * * E R R O * * *
echo               - %msg%
goto fim
:msg
set msg=%msg% %1
:fim
```

Este *script* realiza duas funções principais. A primeira consiste na definição e manutenção de grupos de instâncias a serem gerenciadas. A segunda função realiza, então, o gerenciamento do grupo de instâncias configurado previamente. A utilização deste *script* pode ser efetuada diretamente na console do sistema operacional ou programada através dos recursos do Windows (*At*, *WinAt* ou *Task Scheduler*).

4.3 Customização das regras

O item anterior descreveu todos os requisitos necessários à instalação do módulo de gerenciamento. A partir deste ponto o modelo já apresenta todas as condições para que as regras de gerenciamento passem a ser implementadas.

A maior vantagem deste modelo reside precisamente na definição destas regras customizáveis. Relacionar todas as regras possíveis de serem definidas seria uma tarefa impraticável visto que não existem restrições para a definição de quaisquer tipos de regras. Como exemplificação, entretanto, serão demonstradas algumas regras implementadas no

projeto piloto e que servirão para serem confrontadas com a situação atual descrita no início deste capítulo.

4.3.1 Emissão de alerta baseado na situação atual

Conforme descrito nas características do modelo proposto, este tipo de regra realiza a verificação de valores em visões e/ou tabelas do SGBD. O exemplo abaixo demonstra a verificação e emissão de alerta para objetos críticos, ou seja, que tenham requisitos de alocação superiores à área livre atual:

Criação da regra Critical_Objects_A

```
-- ********************************************************************************
--
--                          CRITICAL_OBJECTS
--
-- ********************************************************************************
```

Identificação da regra (nome da *stored procedure*)

```
CREATE OR REPLACE PROCEDURE CRITICAL_OBJECT_A IS
```

Variáveis de trabalho e cursores

```
   AX_ID_GRC          NUMBER:=012;
   AX_VL_REG          VARCHAR2(200);
   AX_FRE             NUMBER(12);
   AX_CID             INTEGER;
   CURSOR SEGMENTS IS
      SELECT TABLESPACE_NAME, OWNER, SEGMENT_NAME, SEGMENT_TYPE, NEXT_EXTENT
         FROM SYS.DBA_SEGMENTS
            WHERE SEGMENT_TYPE IN ('TABLE','INDEX')
               ORDER BY 1, 2, 3;
   CURSOR FREE_SPACE IS
      SELECT TABLESPACE_NAME, MAX(BYTES) BYTES
         FROM DBA_FREE_SPACE
            GROUP BY TABLESPACE_NAME;
   TYPE TABTYP IS
      TABLE OF DBA_FREE_SPACE.TABLESPACE_NAME%TYPE
      INDEX BY BINARY_INTEGER;
   TAB TABTYP;
   TYPE FRETYP IS
      TABLE OF DBA_FREE_SPACE.BYTES%TYPE
      INDEX BY BINARY_INTEGER;
   FRE FRETYP;
   AX_IN_001          NUMBER(5):=0;
   AX_IN_002          NUMBER(5):=0;
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
```

Verificação dos valores (situação atual ou dados históricos)

```
   FOR FS IN FREE_SPACE LOOP
      AX_IN_001 := AX_IN_001 + 1;
      TAB(AX_IN_001):= FS.TABLESPACE_NAME;
      FRE(AX_IN_001):= FS.BYTES;
   END LOOP;
   FOR SEG IN SEGMENTS LOOP
      AX_FRE:=0;
      FOR AX_IN_002 IN 1..AX_IN_001 LOOP
         IF TAB(AX_IN_002)=SEG.TABLESPACE_NAME THEN
            AX_FRE:= FRE(AX_IN_002);
         END IF;
      END LOOP;
```

Gravação do(s) alerta(s) (caso necessário)

```
      IF SEG.NEXT_EXTENT > AX_FRE THEN
         AX_VL_REG:= LPAD(SEG.NEXT_EXTENT,12,' ');
         INSERT INTO ALERTA VALUES
            (AX_ID_GRC, SYSDATE, SEG.TABLESPACE_NAME, AX_VL_REG, SEG.OWNER || '.' ||
            SEG.SEGMENT_NAME || ' (' || SEG.SEGMENT_TYPE || ') NEXT > FREE ' ||
            AX_FRE, 'N');
```

Ação corretiva (se for o caso)

```
         AX_CID:= DBMS_SQL.OPEN_CURSOR;
         DBMS_SQL.PARSE(AX_CID, 'ALTER ' || SEG.SEGMENT_TYPE || ' "' || SEG.OWNER ||
            '"."' || SEG.SEGMENT_NAME || '" STORAGE (NEXT ' || AX_FRE || ')',
            DBMS_SQL.NATIVE);
         DBMS_SQL.CLOSE_CURSOR(AX_CID);
      END IF;
   END LOOP;
-- Final do processamento
```

Atualização data/hora término última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
```

4.3.2 Coleta de dados históricos

O exemplo abaixo demonstra uma regra que não efetuará a emissão de nenhum alerta. A finalidade deste tipo de regra é armazenar informações na base histórica (genérica) para utilização futura. Neste exemplo são armazenadas as informações de alocação das *tablespaces* do SGBD:

Criação da regra Tablespace_H

```
-- ********************************************************************************
--
--                          TABLESPACE
--
-- ********************************************************************************
```

Identificação da regra (nome da *stored procedure*)

```
CREATE OR REPLACE PROCEDURE TABLESPACE_H IS
```

Variáveis de trabalho e cursores

```
   AX_ID_GRC         NUMBER:=010;
   AX_FREE           NUMBER(12);
   AX_ALOC           NUMBER(12);
   AX_VL_REG         VARCHAR2(200);
   AX_DT_HST         DATE;
   CURSOR TABLESPACE IS
      SELECT TABLESPACE_NAME, SUM(BYTES) TOTAL FROM SYS.DBA_DATA_FILES
         GROUP BY TABLESPACE_NAME;
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Coleta do valor histórico
```

Coleta do(s) valor(es) desejado(s)

```
   SELECT SYSDATE INTO AX_DT_HST FROM DUAL;
   FOR TBS IN TABLESPACE LOOP
      SELECT SUM(BYTES) INTO AX_FREE FROM SYS.DBA_FREE_SPACE
         WHERE TABLESPACE_NAME=TBS.TABLESPACE_NAME;
      IF AX_FREE IS NULL THEN
         AX_FREE:=0;
      END IF;
      SELECT SUM(BYTES) INTO AX_ALOC FROM SYS.DBA_SEGMENTS
         WHERE TABLESPACE_NAME=TBS.TABLESPACE_NAME;
      IF AX_ALOC IS NULL THEN
         AX_ALOC:=0;
      END IF;
```

```
      AX_VL_REG:= LPAD(AX_ALOC,12,' ') || ' - ' || LPAD(AX_FREE,12,' ') || ' - ' ||
         LPAD(TBS.TOTAL,12,' ');
```

Gravação do(s) dado(s) histórico(s)

```
      INSERT INTO HISTORICO VALUES
         (AX_ID_GRC, AX_DT_HST, TBS.TABLESPACE_NAME, AX_VL_REG, 'N');
   END LOOP;
-- Final do processamento
```

Atualização data/hora término última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
```

4.3.3 Emissão de alerta baseado no histórico

Este tipo de regra difere do primeiro porque irá utilizar informações armazenadas na base histórica do módulo de gerenciamento. Estas informações servirão para realizar uma projeção (tendência) do objeto gerenciado, permitindo ações pró-ativas. Neste exemplo são utilizadas as informações armazenadas pela regra do item anterior (alocação das *tablespaces*):

Criação da regra Tablespace_A

```
-- ********************************************************************************
--
--                          TABLESPACE
--
-- ********************************************************************************
```

Identificação da regra (nome da *stored procedure*)

```
CREATE OR REPLACE PROCEDURE TABLESPACE_A
```

Parâmetros de entrada (valores referenciais)

```
   (AX_VL_PRM_001 IN NUMBER, AX_VL_PRM_002 IN NUMBER) IS
```

Variáveis de trabalho e cursores

```
   AX_ID_GRC          NUMBER:=011;
   AX_ID_GRC_REF      NUMBER:=010;
   AX_DT_HST_MAX      DATE;
   AX_DT_HST_REF      VARCHAR2(14);
   AX_NR_AUX          NUMBER:=0;
   AX_NR_DIA          NUMBER:=999999;
   AX_NR_DIA_PRV      NUMBER;
   AX_VL_ALC          NUMBER(12);
   AX_VL_DIA          NUMBER(12);
   CURSOR TABLESPACE_ATUAL IS
      SELECT DE_CMP, DT_HST, VL_REG FROM HISTORICO WHERE ID_GRC=AX_ID_GRC_REF
         AND TO_CHAR(DT_HST,'YYYYMMDDHH24MISS') =
            (SELECT MAX(TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')) FROM HISTORICO
               WHERE ID_GRC=AX_ID_GRC_REF);
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Verificação do valor limite
```

Verificação dos valores (situação atual ou dados históricos)

```
   SELECT MAX(DT_HST) INTO AX_DT_HST_MAX FROM HISTORICO WHERE ID_GRC=AX_ID_GRC_REF;
   SELECT MAX(TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')) INTO AX_DT_HST_REF FROM HISTORICO
      WHERE ID_GRC=AX_ID_GRC_REF
      AND DT_HST < (AX_DT_HST_MAX - AX_VL_PRM_001);
   IF AX_DT_HST_REF IS NULL THEN
      AX_DT_HST_REF:= TO_CHAR(AX_DT_HST_MAX,'YYYYMMDDHH24MISS');
   END IF;
   WHILE AX_NR_DIA = 999999 LOOP
      IF TO_CHAR(AX_DT_HST_MAX - AX_NR_AUX,'YYYYMMDD') =
```

```
            SUBSTR(AX_DT_HST_REF,1,8) THEN
               AX_NR_DIA:= AX_NR_AUX;
         END IF;
         AX_NR_AUX:=AX_NR_AUX + 1;
      END LOOP;
      IF AX_NR_DIA = 0 THEN
         INSERT INTO ALERTA VALUES
            (AX_ID_GRC, SYSDATE, NULL, AX_VL_PRM_001, 'SEM HISTORICO','N');
      ELSE
         FOR TBA IN TABLESPACE_ATUAL LOOP
            SELECT COUNT(*) INTO AX_VL_ALC FROM HISTORICO
               WHERE ID_GRC=AX_ID_GRC_REF AND DE_CMP=TBA.DE_CMP
               AND TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')=AX_DT_HST_REF;
            IF AX_VL_ALC=0 THEN
               AX_VL_ALC:=TO_NUMBER(SUBSTR(TBA.VL_REG,1,12));
            ELSE
               SELECT TO_NUMBER(SUBSTR(VL_REG,1,12)) INTO AX_VL_ALC FROM HISTORICO
                  WHERE ID_GRC=AX_ID_GRC_REF AND DE_CMP=TBA.DE_CMP
                  AND TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')=AX_DT_HST_REF;
            END IF;
Gravação do(s) alerta(s) (caso necessário)
            IF AX_VL_ALC < TO_NUMBER(SUBSTR(TBA.VL_REG,1,12)) THEN
               AX_VL_DIA:= (TO_NUMBER(SUBSTR(TBA.VL_REG,1,12)) - AX_VL_ALC) / AX_NR_DIA;
               AX_NR_DIA_PRV:= TO_NUMBER(SUBSTR(TBA.VL_REG,16,12)) / AX_VL_DIA;
               IF AX_NR_DIA_PRV < AX_VL_PRM_002 THEN
                  INSERT INTO ALERTA VALUES
                     (AX_ID_GRC, SYSDATE, TBA.DE_CMP, LPAD(TRUNC(AX_NR_DIA_PRV),12,' '),
                     'MENOS DE ' || AX_VL_PRM_002 || ' DIAS', 'N');
               END IF;
            END IF;
         END LOOP;
      END IF;
   -- Final do processamento
Atualização data/hora término última execução
      UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
      COMMIT;
   END;
```

4.3.4 Emissão de alerta baseado em critérios de negócio

Este tipo de regra não trata especificamente de valores a serem obtidos nas tabelas e visões do SGBD, mas sim em tabelas e/ou visões de determinado aplicativo. Este tipo de regra ilustra com bastante propriedade a flexibilidade de que o módulo de gerenciamento é dotado. Neste exemplo está sendo descrita a verificação da quantidade de entradas em uma fila de processamento (apontamentos de produção gerados por coletores fabris). O atingimento de uma determinada quantidade deve ser alertado.

Neste exemplo também aparece a aplicação de uma ação corretiva automática (deslocando o registro causador do travamento para o final da fila de processamento). Ações corretivas podem ser empregadas para quaisquer tipo de regras (e não exclusivamente para casos de aplicativos). Mesmo tendo sido tomada esta ação corretiva, o alerta não deve deixar e ser emitido (pois ao administrador será de grande valia tomar conhecimento da freqüência e da quantidade de problemas contornados).

Criação da regra Fila_processamento

```
-- ******************************************************************************
--
--                            FILA PROCESSAMENTO
--
-- ******************************************************************************
```

Identificação da regra (nome da *stored procedure*)

```
CREATE OR REPLACE PROCEDURE FILA_PROCESSAMENTO
```

Parâmetros de entrada (valores referenciais)

```
   (AX_VL_PRM_001 IN VARCHAR2) IS
```

Variáveis de trabalho e cursores

```
   AX_ID_GRC          NUMBER:=045;
   AX_VL_REG          VARCHAR2(200);
   AX_QT_REG          NUMBER(12);
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
```

Verificação dos valores (situação atual ou dados históricos)

```
   SELECT COUNT(*) INTO AX_QT_REG FROM XYZ.FILA_QT;
```

Gravação do(s) alerta(s) (caso necessário)

```
   IF AX_QT_REG > AX_VL_PRM_001 THEN
      AX_VL_REG:= LPAD(AX_QT_REG,12,' ');
      INSERT INTO ALERTA VALUES
         (AX_ID_GRC, SYSDATE, NULL, AX_VL_REG, 'MAIOR QUE ' || AX_VL_PRM_001, 'N');
```

Ação corretiva (se for o caso)

```
      UPDATE XYZ.FILA_QT SET ID_INDICE = ID_INDICE + 900000000000
         WHERE ID_INDICE > 0 AND ROWNUM = 1;
   END IF;
-- Final do processamento
```

Atualização data/hora término última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
```

4.3.5 Emissão de alerta para erros gerais no SGBD

Dependendo do SGBD utilizado, mensagens de erro registradas em estruturas próprias e/ou arquivos do sistema operacional podem ser verificadas. Nesta implementação foi criada uma regra que realiza esta verificação. Através de *feature* que permite o acesso à arquivos foi realizada a varredura das mensagens emitidas pelo SGBD. Um alerta é emitido caso seja encontrado um formato específico de mensagem (que caracteriza um erro).

Criação da regra Erros_SGBD

```
-- ******************************************************************************
--
--                            ERROS SGBD
--
-- ******************************************************************************
```

Identificação da regra (nome da *stored procedure*)

```
CREATE OR REPLACE PROCEDURE ERROS_SGBD
```

Parâmetros de entrada (valores referenciais)

```
   (AX_VL_PRM_001 IN VARCHAR2, AX_VL_PRM_002 IN VARCHAR2) IS
```

Variáveis de trabalho e cursores

```
   AX_ID_GRC          NUMBER:=046;
```

```
   AX_DT_UEX_INI     DATE;
   AX_VL_REG         VARCHAR2(200);
   AX_DT_MSG         DATE;
   AX_FILE           UTL_FILE.FILE_TYPE;
BEGIN
-- Inicio do processamento
```

Atualização data/hora início última execução

```
   SELECT DT_UEX_INI INTO AX_DT_UEX_INI FROM GERENCIAMENTO WHERE ID_GRC = AX_ID_GRC;
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
```

Verificação dos valores (situação atual ou dados históricos)

```
   AX_FILE:= UTL_FILE.FOPEN(AX_VL_PRM_001, AX_VL_PRM_002, 'R');
   LOOP
      UTL_FILE.GET_LINE (AX_FILE,AX_VL_REG);
      IF SUBSTR(AX_VL_REG,21,4) = TO_CHAR(SYSDATE,'YYYY') THEN
         AX_DT_MSG:= TO_DATE(SUBSTR(AX_VL_REG,9,2) || SUBSTR(AX_VL_REG,5,3) ||
            SUBSTR(AX_VL_REG,21,4) || SUBSTR(AX_VL_REG,12,8),'DDMONYYYYHH24:MI:SS');
      END IF;
```

Gravação do(s) alerta(s) (caso necessário)

```
      IF SUBSTR(AX_VL_REG,1,4) = 'ORA-' AND AX_DT_MSG > AX_DT_UEX_INI THEN
         INSERT INTO ALERTA VALUES
            (AX_ID_GRC, SYSDATE, NULL, TO_CHAR(AX_DT_MSG,'YYYY/MM/DD HH24:MI:SS') ||
            ' ' || AX_VL_REG, 'ERRO NO LOG ' || AX_VL_PRM_001 || ' : ' ||
            AX_VL_PRM_002, 'N');
      END IF;
   END LOOP;
UTL_FILE.FCLOSE(AX_FILE);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN;
-- Final do processamento
```

Atualização data/hora término última execução

```
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
```

Outros exemplos de regras customizadas podem ser vistos no Anexo 8.1. Entretanto, não bastariam exemplos para demonstrar a flexibilidade de que este modelo é dotado. Ao imaginar qualquer situação, utilizar valores referenciais dinâmicos e projetar o comportamento de qualquer objeto define-se a enorme capacidade deste modelo de gerenciamento.

4.4 Resumo

Neste capítulo demonstrou-se a viabilidade de implementação deste modelo, utilizando um ambiente real. Foi descrito o ambiente de desenvolvimento utilizado e as etapas necessárias à instalação e customização deste modelo proposto. No próximo capítulo serão discutidos os resultados obtidos com a implementação deste modelo.

5 Resultados da implementação do Modelo Proposto

Este capítulo apresenta os resultados obtidos com o modelo implementado efetuando um comparativo com a situação atual. Este comparativo será focado no modelo geral (visão macro) e também sobre uma situação mais específica de gerenciamento (visão micro).

5.1 Situação Atual

O TNG foi implementado na empresa "X" visando realizar a monitoração do ambiente computacional e acompanhamento de problemas através da utilização de diversas opções (AMO - Inventário de *hardware* e *software*, SSO - Senha única (*Single Sign-on)*, USD - Distribuição de *software*, RCO - Suporte Remoto (*Remote Control*), AHD – Chamados de *Help Desk* e Agentes de monitoração para servidores.

No contexto de servidores, os três aspectos gerais de monitoração são: Sistema Operacional, Banco de dados Oracle e SAP R/3. Após uma análise dos itens monitorados pelo aplicativo concluiu-se que existem alguns *gaps* não cobertos pelo produto, bem como características próprias na empresa que não são atendidas pela forma padrão de gerenciamento do produto. Estes *gaps* se concentram, principalmente, no agente de banco de dados (Oracle).

O fabricante possui alternativas para cobrir estes *gaps* através da implementação de outros produtos ou mesmo o desenvolvimento de módulos a serem integrados ao produto já utilizado. A análise destas alternativas, entretanto, não faz parte do escopo deste estudo de caso. Será realizado um comparativo entre o agente Oracle do TNG e o modelo de gerenciamento proposto nesta dissertação.

5.2 Avaliação dos Resultados

Como forma de demonstrar as potencialidades do modelo proposto e os resultados obtidos, será feito um comparativo entre a situação atual (que utiliza o produto descrito no item anterior) e a alternativa proposta (com a implementação do módulo de gerenciamento apresentado neste trabalho).

Existem diversos módulos que compõem o TNG, sendo que, para servidores, existem três agentes específicos (sistema operacional, banco de dados Oracle e SAP R/3). O módulo de gerenciamento implementado não apresenta alternativa de monitoração para o sistema operacional, pois seu escopo é única e exclusivamente banco de dados SQL (que, no caso desta implementação, é o Oracle).

Apesar do modelo proposto ser específico para SGBD´s que utilizam o padrão SQL, é possível implementar regras para aplicativos que tenham sido instalados utilizando alguma destas bases de dados. Nestes casos, normalmente, as regras implementadas são focadas em critérios de negócios, mas também podem contemplar outros aspectos. Nestas situações será o próprio aplicativo que irá registrar informações particulares à sua forma de implementação ou características de utilização.

Se enquadrariam nesta situação também os aplicativos que utilizam o banco de dados como mero repositório e assumem para si a responsabilidade de certas funções normalmente executadas pelo SGBD (o controle de bloqueio dos registros seria um exemplo). Além disso, alguns aplicativos podem armazenar informações específicas à uma estruturação particular sua. Seria o caso do SAP R/3, pois poderiam ser definidas regras baseadas em informações armazenadas por este ERP, como: desempenho, processos, conexões, cadastro de usuários, etc.

Para efeito de comparação serão abordadas a forma geral de implementação do agente Oracle e um caso específico de objeto gerenciável (*tablespace*) deste agente.

5.2.1 Comparativo geral (módulo de gerenciamento)

A similaridade entre os dois modelos pode ser constatada se considerarmos as definições básicas de gerenciamento, ou seja, a utilização de gerente(s), agentes e objetos gerenciados. Entretanto, o TNG utiliza agentes específicos (para sistema operacional, bancos de dados e aplicativos). Para os bancos de dados, existem agentes distintos do TNG para atender alguns dos SGBD´s existentes. O modelo proposto está focado no gerenciamento de bancos de dados SQL, sendo que existe um agente genérico, ou seja, de utilização aplicável para SGBD´s SQL em geral. Enquanto que no TNG este agente é instalado e executado a partir do sistema operacional, no modelo proposto este agente reside no próprio SGBD.

O TNG utiliza um repositório para armazenar as funções de gerenciamento e os relacionamentos e propriedades dos recursos gerenciados. Esta base de dados é criada especificamente para este fim e pode residir em uma estação de trabalho ou em algum servidor. No modelo proposto, o repositório de dados é criado na própria instância que está sendo gerenciada (sem necessidade de alocação de *hardware* específico para este fim). Além do mais, no modelo proposto, é possível realizar o armazenamento de dados históricos dos objetos gerenciados. Em decorrência desta possibilidade, o modelo proposto também implementa a sumarização e o expurgo (dos dados históricos e dos alertas).

As ações corretivas no TNG são configuradas conforme as mensagens recebidas na console central de eventos. No modelo proposto não existe esta console central, sendo que as ações corretivas são implementadas no próprio agente, ou seja, na identificação de situações que geram alertas já é possível configurar uma ação corretiva apropriada.

Enquanto que o TNG visualiza todo o ambiente em níveis (rede, servidores, agentes e objetos gerenciados) o modelo proposto está focado em instâncias de bancos de dados. É possível fazer a seleção de instâncias que determinado administrador pretenda gerenciar (esta seleção é feita através da composição de grupos de instâncias).

A visualização dos objetos gerenciados no TNG é feita através de uma interface gráfica (Mapa 2-D). Um padrão de cores permite a navegação pelos níveis de visualização (até que se atinja o objeto gerenciado que apresenta algum aviso ou alerta crítico). No modelo proposto a visualização dos alertas e históricos pode ser feita com as ferramentas disponibilizadas por cada SGBD (ou terceiros), considerando-se que esta visualização somente será necessária quando da notificação de novos alertas (conforme descrito anteriormente).

O TNG utiliza dois níveis para realizar a monitoração de determinado recurso gerenciado. Estes níveis são expressos em percentuais e indicam a necessidade de emissão de um aviso ou de um alerta crítico. No modelo proposto é possível fazer a implementação da mesma forma. Entretanto, também é possível trabalhar com valores de outros tipos que não apenas percentuais, ou seja, o comparativo pode ser feito com números, datas e mesmo caracteres (se for necessário). Valores que, inclusive, podem ser dinâmicos (passados através de parâmetros).

As regras disponíveis para o agente Oracle do TNG são: *Tablespaces*, *Data Disks*, *Data Files*, *Licensing* (*Users*/*Sessions*), *Tables* e *Sequences.* Excetuando-se a regra *Data Disks*, todas as outras estão baseadas em informações contidas em tabelas e visões do próprio SGBD e, portanto, passíveis de serem implementadas pelo modelo proposto. Além disso, é possível criar regras para outros objetos que não apenas os contidos na lista do TNG.

O item acima comenta que a regra *Data Disks* não pode ser implementada no modelo proposto pelo fato de suas informações não estarem contidas em tabelas e visões do SGBD. Esta informação precisa ser extraída do sistema operacional e o modelo proposto está incorporado ao SGBD (que não dispõem destas informações). Este objeto gerenciado, entretanto, não retrata uma entidade do SGBD e deveria estar incorporado no agente para sistema operacional do TNG. Esta impossibilidade está caracterizada por uma interpretação do produto TNG (que efetua a monitoração dos discos que contenham arquivos do banco de dados, independentemente de conterem outros tipos de arquivos ou não). Portanto, esta impossibilidade não é causada por uma limitação do modelo proposto (que abrange qualquer objeto contido no SGBD).

Havendo a necessidade de contornar alguma situação, ou seja, trabalhar com objetos que caracterizem uma exceção à regra implementada pelo TNG, não é possível acrescentar filtros ao critério de aplicação deste regra. No modelo proposto, a aplicação de uma *where-*

condition implementa o filtro necessário a esta necessidade (através de condições das mais simples até as mais elaboradas, se for necessário).

As avaliações realizadas pelo TNG são baseadas na situação atual dos objetos gerenciados. No modelo proposto esta avaliação pode ser realizada da mesma forma, mas também pode ser baseada em informações históricas do objeto gerenciado, efetuando-se uma projeção de tendência deste objeto. Esta projeção pode trabalhar com períodos que vão desde minutos até anos (utilizando-se o que for mais apropriado para cada caso).

Existe uma console de eventos no TNG que faz a recepção dos *traps* enviados pelos agentes. As mensagens recebidas poderão desencadear algumas ações como o envio de *e-mail* para os administradores (no caso de categorias específicas de mensagens). Quando ocorrer a repetição de um mesmo erro com uma alta freqüência, serão enviados *e-mails* aos administradores na mesma quantidade em que forem emitidos alertas na console. No modelo proposto existe um processo de verificação de novos alertas que notifica o administrador deste evento, sem poluir a estação de gerenciamento com todos os alertas gerados. Ele, por sua vez, fará a consulta aos alertas existentes utilizando-se de ferramentas disponibilizadas por cada SGBD (com critérios que permitam a seleção dos mesmos através de agrupamentos ou restrições). O administrador receberá, também, conforme sua programação, um resumo do *status* das instâncias sob sua responsabilidade.

Para as regras *Tables* e *Sequences* não ocorre o reconhecimento automático pelo TNG (como é caso das outras regras). Neste caso, para realizar o monitoramento destes tipos de objetos torna-se necessário incluir, manual e individualmente, cada uma das tabelas ou seqüências desejadas. No modelo proposto é possível incluir todos os objetos existentes ou mesmo restringir esta lista através de critérios contidos na própria regra.

Quando o TNG faz o reconhecimento automático dos objetos existentes também assume valores referenciais padrões (*defaults*). Se, para determinados objetos, pretende-se utilizar outros valores referenciais (maiores ou menores que os propostos) será preciso alterar, manual e individualmente, cada um deles. Se determinado objeto apresenta um comportamento ou características diferenciadas dos demais, pode ser necessário elevar os valores referenciais (para evitar alertas constantes não condizentes com a realidade) ou simplesmente deixar de monitorar o objeto diferenciado. No modelo proposto esta restrição pode ser efetuada na codificação da própria regra, podendo, inclusive, utilizar valores referenciais dinâmicos (recebidos através de parâmetros na chamada da *stored procedure*).

Comparativo Geral (Resumo)

Unicenter Tng	**Modelo Proposto**
Agentes específicos para DB2, Informix, Ingres, Oracle, SQL Server e Sybase (instalados no S.O.)	Agente SQL (genérico) baseado em *stored procedures* (armazenadas no SGBD)
Repositório (funções, gerenciamento, relacionamentos e propriedades) criado em HW específico	Repositório de dados criado na instância gerenciada (permite coleta de dados históricos)
Ações corretivas configuradas na console central de eventos	Ações corretivas implementadas no próprio agente (Alertas)
Visualização geral em níveis	Visualização por grupos de instâncias
Mapa 2-D para objetos (utilização de padrão de cores)	Ferramentas do SGBD quando ocorrerem notificações
Dois níveis percentuais para monitoração	Qualquer tipo de dado (parâmetro)
Regras pré-definidas para lista finita de objetos	Regras customizáveis para quaisquer objetos (critérios de negócio, inclusive)
Inexistência de filtros	Utilização de *where-condition* para implementação de filtros
Avaliação da situação atual	Avaliação das informações históricas
Console central de eventos recebe todos *traps* enviados pelos agentes	Verificação de novos alertas
Identificação de problemas baseados na recepção de erros	Resumo da situação do módulo de gerenciamento em cada instância
Recursos passíveis de configuração requerem inserção manual/unitária	Inclusão/exclusão de objetos conforme critérios da própria regra
Reconhecimento automático com valores *default* (alteração manual)	Valores referenciais dinâmicos (recebidos através de parâmetros)

5.2.2 Comparativo específico (regras para *tablespaces*)

Após o comparativo geral do módulo de gerenciamento, realizado no item anterior, a comparação será mais específica. Para realizar esta comparação optou-se por um tipo de objeto bastante significativo na administração de bancos de dados Oracle. Tratam-se das *tablespaces*, que são entidades lógicas compostas por um ou mais arquivos físicos residentes no sistema operacional. Cada *tablespace* irá conter segmentos específicos alocados em extensões.

Além das *tablespaces* mais comuns, utilizadas para armazenar as tabelas e os índices dos aplicativos que utilizam o banco de dados, existem outras três que possuem características diferenciadas. A primeira serve como repositório para o dicionário de dados do SGBD e denomina-se SYSTEM. A Segunda, denominada TEMP, armazena segmentos temporários (utilizados para cláusulas *order by* ou criação de índices). RBS é a última tablespace e armazena segmentos de *rollback* (utilizados para garantir a atomicidade do banco de dados).

O gerenciamento do TNG para *Tablespaces* segue o mesmo padrão dos outros objetos, ou seja, para cada uma delas existirão dois valores percentuais para indicar a necessidade de emissão de um aviso ou de um alerta crítico. Para estes percentuais são assumidos os valores *default* de 70% e 80%, respectivamente, conforme demonstrado na figura 5.1.

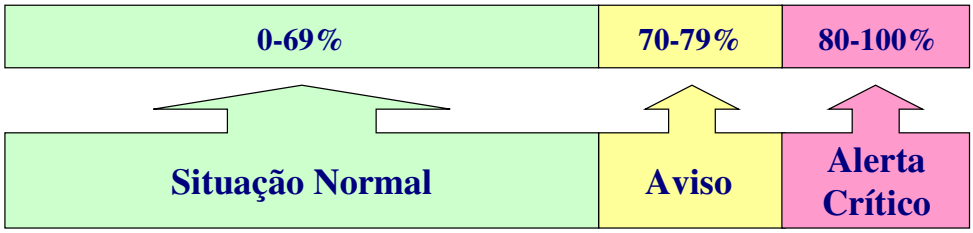

Figura 5.1 – Percentuais *default* para tablespaces

Assumir que 80% de utilização é a margem segura para administração da área de um banco de dados implica em assumir que, pelo menos, 20% do espaço alocado não está sendo utilizado e, consequentemente, poderá estar sendo desperdiçado.

Elevar este percentual para 90% ou mais pode favorecer situações em que a área livre seja esgotada muito rapidamente, não permitindo ações preventivas em tempo hábil.

Identificar o comportamento de cada uma das *tablespaces* e utilizar percentuais mais realistas para cada caso é uma tarefa desgastante, mesmo porque o comportamento de determinada *tablespace* pode variar ao longo do tempo (obrigando revisões constantes).

Excluir determinadas *tablespaces* da lista de objetos gerenciados por sempre apresentarem ocupação máxima é uma ação que fere o objetivo das atividades de um administrador, pois haverá uma entidade que não está sendo gerenciada.

As considerações acima mostram as possibilidades de administração das *tablespaces* de um banco de dados através do agente Oracle do TNG e refletem o que foi observado em um ambiente real. A seguir será apresentada uma alternativa de gerenciamento adotada na implementação do modelo proposto. As regras que serão comentadas podem ser encontradas no Anexo 8.1.

Nenhuma regra baseada em percentual de utilização das *tablespaces* foi implementada por concluir-se que esta abordagem não condiz com as possibilidades reais defrontadas. Até poderia ser implementada como forma adicional de gerenciamento (complementando as medidas que serão descritas na seqüência), mas pretende-se demonstrar que as regras efetivamente implementadas são suficientes e, principalmente, eficazes para este gerenciamento.

A figura 5.2 exemplifica o comportamento de duas *tablespaces*. No instante T1, se for considerado o percentual de utilização tem-se a *tablespace* A em estado crítico. Contudo, se for verificado o crescimento real destas *tablespaces*, constata-se que é a *tablespace* B (no instante T2) que efetivamente atingiu um alto grau de criticidade.

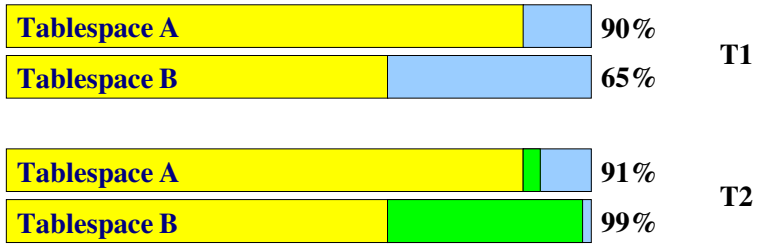

Figura 5.2 – Exemplo de comportamento de *tablespaces*

A primeira regra customizada intitula-se Tablespace_H e corresponde ao processo de coleta de dados históricos das *tablespaces*. Esta coleta pode ser realizada de acordo com as projeções que se pretendem fazer, ou seja, se houver coletas diárias poderão ser realizadas projeções para o crescimento previsto para os próximos dias, semanas, meses ou anos.

O armazenamento dos dados históricos realizado pela regra Tablespace_H serve como base de informação para a regra Tablespace_A que efetua projeções de crescimento e estimativa (em dias) de quando ocorrerá o provável esgotamento da área livre. Esta regra recebe dois valores através da passagem de parâmetros. O primeiro expressa a quantidade de dias (intervalo) para realizar a projeção. Exemplo: 90 indica que serão comparados os valores mais recentes com os valores coletados há pelo menos 90 dias atrás (traduzindo-se na média trimestral).

O segundo parâmetro recebido por esta regra expressa o valor limite (em dias) que determina a emissão de uma alerta. Exemplo: assumindo que 90 e 30 sejam os dois parâmetros recebidos pela regra. Neste caso, baseado na média de crescimento do trimestre, será emitido um alerta para cada *tablespace* que, em menos de 30 dias, tenha a tendência de esgotar toda a sua área livre.

Com estas duas regras obtém-se um gerenciamento bastante eficaz das *tablespaces*, pois não será preciso realizar super-alocações para aquelas que tenham uma média de crescimento baixa. Para o caso inverso, ou seja, *tablespaces* que tenham uma média de crescimento elevada, será possível alocar, antecipadamente, área suficiente para garantir o seu crescimento projetado para determinado período.

Entretanto, existem detalhes que ainda podem ser considerados para melhorar ainda mais este gerenciamento. Conforme comentado, cada *tablespace* estará comportando

segmentos alocados em extensões. Cada vez que um determinado segmento necessite mais área (para comportar a inclusão de novos registros, por exemplo) será preciso alocar uma nova extensão para o mesmo. Em algumas situações, pode existir um total de área livre na *tablespace* aparentemente suficiente para um longo período. Contudo, como esta área livre também é composta por extensões, não há garantias de que ela seja contígua (a figura 5.3 demonstra a fragmentação de segmentos em uma *tablespace*). Para que um determinado segmento obtenha uma extensão com êxito será preciso que exista uma área livre contígua maior ou igual ao tamanho desta extensão.

Figura 5.3 – Fragmentação de segmentos em uma *tablespace*

Portanto, para controlar estas situações, foi implementada a regra denominada Critical_Objects_A. Esta regra se encarrega de verificar os requisitos de extensão dos segmentos de cada *tablespace*, emitindo alertas quando não existe área contígua suficiente. Esta regra também é dotada de uma ação corretiva automática, ou seja, ela se encarrega de emitir o alerta, mas também realiza o ajuste das extensões dos objetos de acordo com a maior área livre contígua de cada *tablespace*.

O agente Oracle do TNG disponibiliza uma regra denominada *Tables* destinada ao gerenciamento de tabelas. O objetivo desta regra, entretanto, é verificar o percentual de extensões utilizadas por uma tabela em relação ao máximo permitido (diferente do propósito da regra customizada descrita acima). Na versão do SGBD utilizado é possível especificar como *unlimited* esta quantidade de extensões, não havendo razão para adotar tal regra (mesmo porque seria necessário cadastrar todas as tabelas desejadas manualmente).

Na regra de *Tablespaces* o TNG apresenta o percentual de blocos livres localizados entre os blocos de dados, contudo, não informa se esta fragmentação compromete ou não a alocação de extensões para os objetos. Adotar um determinado percentual como crítico faria com que fosse realizado o aumento da área alocada (por precaução), mas que aumentaria a área livre não ocupada (desperdiçando, eventualmente, espaço em disco).

Para concluir o gerenciamento das *tablespaces* ainda foram implementadas duas regras denominadas Rollback_Used_Ublk_H e Rollback_Used_Ublk_A, realizando o trabalho de coleta de dados históricos e emissão de alertas, respectivamente. Estas regras estão focadas nos segmentos de *rollback* (alocados em *tablespace* específica). A alocação destes segmentos pode ser realizada através da ocupação total da área da *tablespace*, contudo, não significando que a mesma esteja com problemas (nem todos segmentos

estarão, necessariamente, sendo utilizados). A figura 5.4 demonstra esta situação. Para saber a ocupação dos segmentos de *rollback* estas regras realizam o acompanhamento da utilização real dos blocos de *undo* de todas as transações em paralelo emitindo um alerta quando o valor de utilização ultrapasse o valor de referência (informado através de parâmetro).

Figura 5.4 – Utilização dos segmentos de *rollback*

5.3 Resumo

Neste capítulo foram apresentados os resultados obtidos com o modelo implementado através de um comparativo com a situação atual. Este comparativo apresentou uma visão macro do modelo e também focou uma situação mais específica de gerenciamento (visão micro). Esta apresentação encerra o tema proposto para este trabalho, sendo que na seqüência serão emitidas as conclusões finais desta dissertação.

6 Conclusões

O objetivo geral deste trabalho foi desenvolver um modelo de gerenciamento para banco de dados SQL. A principal contribuição científica deste trabalho reside na definição de um modelo que permite a inclusão de regras de gerenciamento customizáveis.

Para viabilizar este modelo genérico foram estudadas as características gerais dos bancos de dados SQL. A utilização de *stored procedures* foi a opção natural para este modelo, pois apresentam benefícios significativos para o desempenho (minimizando o *overhead* causado pelo módulo de gerenciamento). A utilização da linguagem nativa do SGBD (SQL) também trouxe facilidade na compreensão e utilização do modelo.

Não faz parte do escopo deste trabalho a avaliação de produtos de gerenciamento disponíveis no mercado. Entretanto, foi descrito o modelo adotado pelo Unicenter TNG com o intuito de permitir comparações entre os modelos.

O modelo genérico de gerenciamento de redes foi utilizado como referência, sendo que o módulo proposto foi adequado à este padrão. Além das regras customizáveis, foram definidas as demais características do modelo proposto (Programação Periódica, Histórico Genérico, Sumarização, Expurgo, Notificações e Ações Corretivas).

O modelo de dados foi definido e, apesar de enxuto, mostrou-se adequado à utilização do módulo genérico, ou seja, na implementação do modelo todas as necessidades foram atendidas pelas tabelas Gerenciamento, Histórico, Alerta e Monitor (não foi identificada a necessidade de nenhuma outra entidade).

A implementação do modelo proposto foi realizada em um ambiente que utiliza o SGBD Oracle. Esta implementação serviu como forma de validação do modelo, sendo que, conforme demonstrado no capítulo 4, a instalação e customização do módulo de gerenciamento foram realizadas e concluídas com êxito.

Os resultados desta implementação permitiram um comparativo com o Unicenter TNG. Desconsiderando-se as diferenças de concepção entre os modelos e ressaltando apenas o quesito das regras de gerenciamento pode-se concluir que o modelo proposto atendeu as expectativas, mostrando grande eficiência. O comparativo apresentado no capítulo 5 demonstrou que a composição de regras customizáveis dota o modelo proposto de uma flexibilidade extrema.

Como sugestão para trabalhos futuros indica-se a implementação do modelo proposto em outros SGBD´s. Estas implementações deverão corroborar a validade deste modelo. Como recomendação para este trabalho poderiam ser utilizados o SQL Server (forte concorrente do Oracle) e o MySql (líder entre os SGBD´s de código-fonte aberto).

Outro ponto que poderia ser considerado para o modelo proposto é a definição de inter-relacionamentos entre as instâncias gerenciadas. O modelo proposto dota cada instância de autonomia, mas poderiam ser obtidos benefícios caso o trabalho de gerenciamento fosse realizado de forma colaborativa. Um exemplo seria a implementação de sincronismo de regras, ou seja, a inclusão de uma nova *stored procedure* poderia ser efetuada em qualquer instância e o modelo garantiria a propagação (replicação) para as demais instâncias gerenciadas.

Ainda focando o inter-relacionamento das instâncias, outra possibilidade seria a utilização, de forma cruzada, do repositório de dados. No caso de alguma falha, seria possível consultar os dados relativos à esta instância (que poderiam auxiliar na identificação da causa da falha). Em ambos os casos é preciso avaliar com critério a inclusão destas novas características, preservando o modelo genérico. Este cuidado se faz necessário pois a utilização de *features* específicas de determinados SGBD´s poderia agregar facilidades ao modelo, mas descaracterizariam o mesmo (criando funcionalidades para um conjunto restrito de SGBD´s).

A principal vantagem do modelo proposto é a possibilidade de definição de regras customizáveis, agregando grande flexibilidade na composição de novas regras de gerenciamento (qualquer situação ou necessidade específica pode ser incorporada à solução com extrema facilidade). Esta característica permite que sejam criadas tantas regras quantas forem necessárias e, principalmente, viabiliza ajustes na verificação dos objetos gerenciados. Desta forma, alterações no gerenciamento decorrentes de novos critérios, alteração da versão do SGBD, novas situações identificadas ou restrições de determinada natureza são facilmente implementadas.

A adoção de uma estrutura genérica para os dados históricos traz como vantagens: utilização de um repositório único para o armazenamento dos dados (independentemente da quantidade de regras criadas); dados de diferentes formatos em campo único (através de funções de conversão); diversos valores em um único registro, mesmo utilizando formatos diferentes; armazenamento de entradas unitárias ou múltiplas para os objetos gerenciados.

A parametrização é um dos pontos que distinguem o modelo, ou seja, permitem flexibilidade na utilização do módulo de gerenciamento. Períodos e/ou formas de sumarização, prazo de retenção de dados, valores referenciais para as regras de gerenciamento, intervalos para verificação (monitoração) e grupos de instâncias gerenciadas são exemplos desta parametrização. Estes casos, sendo definidos por parâmetros são, por conseqüência, facilmente ajustados/alterados de acordo com as necessidades do ambiente gerenciado.

A definição de critérios para a sumarização e expurgo dos dados (Histórico e Alerta) garante a utilização otimizada de recursos de armazenamento. A independência do

fornecedor, portabilidade e utilização de um código-fonte aberto também são características positivas do modelo proposto.

Conforme comentado inicialmente, o estudo detalhado de produtos de gerenciamento não faz parte do escopo deste trabalho, entretanto, durante as pesquisas realizadas foi possível tomar conhecimento de algumas alternativas existentes. Considerando-se as vantagens apresentadas pelo modelo proposto é possível afirmar que poucas soluções pesquisadas apresentaram algumas destas vantagens. Não foi encontrada nenhuma alternativa de solução que apresentasse todas as vantagens descritas neste modelo proposto.

Para finalizar, destaca-se que este modelo proposto foi concebido observando-se um formato genérico (aplicável para qualquer SGBD SQL) e conclui-se que o objetivo principal (regras de gerenciamento customizáveis) foi alcançado, conforme observado nos resultados da implementação deste modelo.

7 Bibliografia

[01] SQL Server - Panorama. Disponível em: <http://www.provue.com/proVUE/Fact_SQLServer.html>. Acesso em: 26 de junho de 2002.

[02] Database Manager – SQL History/Overview. Disponível em: <http://www.itworld.com/nl/db_mgr/05142001>. Acesso em: 26 de junho de 2002.

[03] SQL: The Standard and the Language. Disponível em: <http://www.opengroup.org/public/tech/datam/sql.htm>. Acesso em: 26 de junho de 2002.

[04] Database Manager - Introduction to SQL. Disponível em: <http://www.itworld.com/nl/db_mgr/05282001>. Acesso em: 26 de junho de 2002.

[05] Writing a Stored Procedure. Disponível em: <http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=127&lngWId-5>. Acesso em: 21 de novembro de 2002.

[06] Stored Procedures with SQL Server. Disponível em: <http://databases.about.com/library/weekly/aa042201ªhtm>. Acesso em: 21 de novembro de 2002.

[07] Stored Procedures explainded. Disponível em: <http://ww7.boulder.ibm.com/vajdoc/vahwebx.exe/en_US/vj32/Extract/0/spbuild/concepts/cswhtrsp.htm>. Acesso em: 21 de novembro de 2002.

[08] Introduction to Stored Procedures. Disponível em: <http://www.vbip.com/books/1861003064/chapter_3064_01.asp>. Acesso em: 21 de novembro de 2002.

[09] Oracle8i SQL Reference. Release 8.1.5. Part Number A67779-01.

[10] Projeto Orientado I. Disponível em: <http://www.geocities.com/Silicom Valley/Vista/5635/sumario.html>. Acesso em: 29 de junho de 2002.

[11] Gerente: Estado da arte e justificativas. Disponível em: <http://www.inf.puc-rio.br/~michael/gerente/P01.htm>. Acesso em: 29 de junho de 2002.

[12] LanTimes Brasil - A geografia do gerenciamento. Editora Rever Ltda. Nov/1998.

[13] Unicenter TNG: Concepts Guide – Release 2.1. Computer Associates.

[14] CA Common Services Architecture Overview. Disponível em: <www3.ca.com/Files/WhitePapers/ca_common_services_wp.pdf>. Acesso em: 02 de dezembro de 2002.

[15] Unicenter TNG 2.4 Manuals: Using the Oracle Agent. Disponível em: <http://support.ca.com/premium/unicenter/manuals/24Doc/agenttech/useora.pdf>. Acesso em: 26 de novembro de 2002.

[16] Unicenter TNG 2.4 Manuals: Using the DB2 Agent. Disponível em: <http://support.ca.com/premium/unicenter/manuals/24Doc/agenttech/db2agent.pdf> Acesso em: 26 de novembro de 2002.

[17] Unicenter TNG 2.4 Manuals: Using the Informix Database Agent. Disponível em: <http://support.ca.com/premium/unicenter/manuals/24Doc/agenttech/usginfmx.pdf> Acesso em: 02 de dezembro de 2002.

[18] Unicenter TNG 2.4 Manuals: Using the Ingres Agent. Disponível em: <http://support.ca.com/premium/unicenter/manuals/24Doc/agenttech/useing.pdf>. Acesso em: 26 de novembro de 2002.

[19] Unicenter TNG 2.4 Manuals: Using the SQL Server Agent. Disponível em: <http://support.ca.com/premium/unicenter/manuals/24Doc/agenttech/usingsql.pdf>. Acesso em: 26 de novembro de 2002.

[20] Unicenter TNG 2.4 Manuals: Using the Sybase SQL Server MIBMUX Agent. Disponível em: <http://support.ca.com/premium/unicenter/manuals/24Doc/agenttech/usesyb.pdf>. Acesso em: 02 de dezembro de 2002.

[21] Unicenter TNG SDK. Disponível em: <http://www.cai.com/develop/tngsdk>. Acesso em: 02 de dezembro de 2002.

[22] Unicenter TNG 2.2 Manuals: CA SDK Programmer Guide. Disponível em: <http://support.ca.com/tng22manuals/sdkprog>. Acesso em: 02 de dezembro de 2002.

[23] Oracle8i Administrator´s Guide. Release 8.1.5. Part Number A67772-01.

[24] Oracle8i Application Developer´s Guide – Advanced Queuing. Release 8.1.5. Part Number A68005-01.

[25] Oracle8i Application Developer's Guide – Fundamentals. Release 8.1.5. Part Number A68003-01.

[26] Oracle8i Concepts. Release 8.1.5. Part Number A67781-01.

[27] Oracle8i Generic Documentation Master Index. Release 8.1.5. Part Number A68826-01.

[28] Oracle8i Reference. Release 8.1.5. Part Number A67790-01.

[29] Oracle8i Supplied Packages Reference. Release 8.1.5. Part Number A68001-01.

[30] Oracle8i Tuning. Release 8.1.5. Part Number A67775-01.

[31] Oracle Enterprise Manager Configuration Guide. Release 2.2. Part Number A85247-01.

[32] Oracle Enterprise Manager FAQ. Disponível em: <http://www.orafaq.com/faqoem.htm>. Acesso em: 15 de maio de 2002.

[33] Oracle Intelligent Agent Users Guide. Release 8.1.5. Part Number A67825-01.

[34] Oracle SNMP Support Reference Guide. Release 8.1.7. Part Number A85249-01.

[35] Dba Support: Oracle Scripts. Disponível em: <http://dbasupport.com/Oracle/scripts>. Acesso em: 22 de agosto de 2002.

[36] ROSE, M.; MCCLOGHRIE, K. RFC 1155 Structure and identification of management information for TCP/IP-based Internets. May, 1990.

[37] MCCLOGHRIE, K.; ROSE, M. RFC 1156 Management information base for network management of TCP/IP-based internets. May, 1990.

[38] BROWER, D.; PURVY, B.; SINYKIN, M.; SMITH, J. RFC 1697 Relational Database Management System (RDBMS) Management Information Base (MIB) using SMIv2. Aug, 1994.

[39] MCCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. RFC 2578 Structure of management information version 2 (SNMPv2). Apr., 1999.

[40] MCCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. RFC 2580 Conformance statements for SMIv2.Apr., 1999.

8 Anexos

8.1 Exemplos de regras customizadas (*Stored Procedures*)

```
-- ******************************************************************************
--
--                          ROLLBACK_USED_UBLK
--
-- ******************************************************************************
CREATE OR REPLACE PROCEDURE ROLLBACK_USED_UBLK_H IS
   AX_ID_GRC         NUMBER:=001;
   AX_USED_UBLK      NUMBER(8);
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Coleta do valor histórico
   SELECT SUM(USED_UBLK)
      INTO AX_USED_UBLK
      FROM V$TRANSACTION;
   IF AX_USED_UBLK IS NULL THEN
      AX_USED_UBLK:=0;
   END IF;
   INSERT INTO HISTORICO VALUES
      (AX_ID_GRC, SYSDATE, NULL, LPAD(AX_USED_UBLK,8,' '), 'N');
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE ROLLBACK_USED_UBLK_A (AX_VL_PRM_001 IN NUMBER) IS
   AX_ID_GRC         NUMBER:=002;
   AX_ID_GRC_REF     NUMBER:=001;
   AX_DT_UEX_INI     DATE;
   AX_USED_UBLK      NUMBER;
   AX_VL_REG         VARCHAR2(200);
BEGIN
-- Inicio do processamento
   SELECT DT_UEX_INI INTO AX_DT_UEX_INI FROM GERENCIAMENTO WHERE ID_GRC = AX_ID_GRC;
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Verificação do valor limite
   SELECT MAX(VL_REG) INTO AX_VL_REG FROM HISTORICO
      WHERE ID_GRC = AX_ID_GRC_REF AND DT_HST > AX_DT_UEX_INI;
   AX_USED_UBLK:= TO_NUMBER(SUBSTR(AX_VL_REG,1,8));
   IF AX_USED_UBLK > AX_VL_PRM_001 THEN
      INSERT INTO ALERTA VALUES (AX_ID_GRC, SYSDATE, NULL, AX_VL_REG, 'MAIOR QUE ' ||
      AX_VL_PRM_001, 'N');
   END IF;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ******************************************************************************
--
--                          LIBRARY_CACHE
--
-- ******************************************************************************
CREATE OR REPLACE PROCEDURE LIBRARY_CACHE_H IS
   AX_ID_GRC         NUMBER:=003;
   AX_PC_CACHE       NUMBER(6,3);
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Coleta do valor histórico
   SELECT (SUM(RELOADS)/SUM(PINS)*100) INTO AX_PC_CACHE FROM V$LIBRARYCACHE;
   IF AX_PC_CACHE IS NULL THEN
      AX_PC_CACHE:=0;
```

```
   END IF;
   INSERT INTO HISTORICO VALUES
      (AX_ID_GRC, SYSDATE, NULL, LPAD(AX_PC_CACHE,7,' '), 'N');
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE LIBRARY_CACHE_A (AX_VL_PRM_001 IN NUMBER) IS
   AX_ID_GRC         NUMBER:=004;
   AX_ID_GRC_REF     NUMBER:=003;
   AX_DT_UEX_INI     DATE;
   AX_PC_CACHE       NUMBER(6,3);
   AX_VL_REG         VARCHAR2(200);
BEGIN
-- Inicio do processamento
   SELECT DT_UEX_INI INTO AX_DT_UEX_INI FROM GERENCIAMENTO WHERE ID_GRC = AX_ID_GRC;
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Verificação do valor limite
   SELECT MAX(VL_REG) INTO AX_VL_REG FROM HISTORICO
      WHERE ID_GRC = AX_ID_GRC_REF AND DT_HST > AX_DT_UEX_INI;
   AX_PC_CACHE:= TO_NUMBER(SUBSTR(AX_VL_REG,1,7));
   IF AX_PC_CACHE > AX_VL_PRM_001 THEN
      INSERT INTO ALERTA VALUES (AX_ID_GRC, SYSDATE, NULL, AX_VL_REG, 'MAIOR QUE ' ||
      AX_VL_PRM_001, 'N');
   END IF;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ********************************************************************************
--
--                          DICTIONARY_CACHE
--
-- ********************************************************************************
CREATE OR REPLACE PROCEDURE DICTIONARY_CACHE_H IS
   AX_ID_GRC         NUMBER:=005;
   AX_PC_CACHE       NUMBER(6,3);
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Coleta do valor histórico
   SELECT (SUM(GETMISSES)/SUM(GETS)*100) INTO AX_PC_CACHE FROM V$ROWCACHE;
   IF AX_PC_CACHE IS NULL THEN
      AX_PC_CACHE:=0;
   END IF;
   INSERT INTO HISTORICO VALUES
      (AX_ID_GRC, SYSDATE, NULL, LPAD(AX_PC_CACHE,7,' '), 'N');
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE DICTIONARY_CACHE_A (AX_VL_PRM_001 IN NUMBER) IS
   AX_ID_GRC         NUMBER:=006;
   AX_ID_GRC_REF     NUMBER:=005;
   AX_DT_UEX_INI     DATE;
   AX_PC_CACHE       NUMBER(6,3);
   AX_VL_REG         VARCHAR2(200);
BEGIN
-- Inicio do processamento
   SELECT DT_UEX_INI INTO AX_DT_UEX_INI FROM GERENCIAMENTO WHERE ID_GRC = AX_ID_GRC;
    UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Verificação do valor limite
   SELECT MAX(VL_REG) INTO AX_VL_REG FROM HISTORICO
      WHERE ID_GRC = AX_ID_GRC_REF AND DT_HST > AX_DT_UEX_INI;
   AX_PC_CACHE:= TO_NUMBER(SUBSTR(AX_VL_REG,1,7));
   IF AX_PC_CACHE > AX_VL_PRM_001 THEN
      INSERT INTO ALERTA VALUES (AX_ID_GRC, SYSDATE, NULL, AX_VL_REG, 'MAIOR QUE ' ||
      AX_VL_PRM_001, 'N');
```

```
   END IF;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ******************************************************************************
--
--                          BUFFER_CACHE
--
-- ******************************************************************************
CREATE OR REPLACE PROCEDURE BUFFER_CACHE_H IS
   AX_ID_GRC         NUMBER:=007;
   AX_PC_CACHE       NUMBER(6,3);
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Coleta do valor histórico
   SELECT (1 - SUM(DECODE(NAME,'physical reads', value, 0)) /
      (SUM(DECODE(NAME,'db block gets', value, 0)) +
      SUM(DECODE(NAME,'consistent gets', value, 0))))*100 INTO AX_PC_CACHE
         FROM SYS.V_$SYSSTAT WHERE NAME IN
            ('consistent gets','db block gets', 'physical reads');
   IF AX_PC_CACHE IS NULL THEN
      AX_PC_CACHE:=0;
   END IF;
   INSERT INTO HISTORICO VALUES
      (AX_ID_GRC, SYSDATE, NULL, LPAD(AX_PC_CACHE,7,' '), 'N');
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE BUFFER_CACHE_A (AX_VL_PRM_001 IN NUMBER) IS
   AX_ID_GRC         NUMBER:=008;
   AX_ID_GRC_REF     NUMBER:=007;
   AX_DT_UEX_INI     DATE;
   AX_PC_CACHE       NUMBER(6,3);
   AX_VL_REG         VARCHAR2(200);
BEGIN
-- Inicio do processamento
   SELECT DT_UEX_INI INTO AX_DT_UEX_INI FROM GERENCIAMENTO WHERE ID_GRC = AX_ID_GRC;
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Verificação do valor limite
   SELECT MAX(VL_REG) INTO AX_VL_REG FROM HISTORICO
      WHERE ID_GRC = AX_ID_GRC_REF AND DT_HST > AX_DT_UEX_INI;
   AX_PC_CACHE:= TO_NUMBER(SUBSTR(AX_VL_REG,1,7));
   IF AX_PC_CACHE < AX_VL_PRM_001 THEN
      INSERT INTO ALERTA VALUES (AX_ID_GRC, SYSDATE, NULL, AX_VL_REG, 'MENOR QUE ' ||
         AX_VL_PRM_001, 'N');
   END IF;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ******************************************************************************
--
--                          EMPTY_DATA_FILE
--
-- ******************************************************************************
CREATE OR REPLACE PROCEDURE EMPTY_DATA_FILE_A IS
   AX_ID_GRC         NUMBER:=009;
   AX_VL_REG         VARCHAR2(200);
   CURSOR EMPTY_DATA_FILE IS
      SELECT TABLESPACE_NAME, FILE_ID, BYTES, FILE_NAME FROM SYS.DBA_DATA_FILES
      WHERE FILE_ID NOT IN (SELECT DISTINCT FILE_ID FROM SYS.DBA_EXTENTS)
      ORDER BY 1, 2;
BEGIN
-- Inicio do processamento
```

```
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
   FOR EDF IN EMPTY_DATA_FILE LOOP
      AX_VL_REG:= LPAD(EDF.BYTES,12,' ');
      INSERT INTO ALERTA VALUES (AX_ID_GRC, SYSDATE, EDF.TABLESPACE_NAME, AX_VL_REG,
         EDF.FILE_NAME || ' (' || EDF.FILE_ID || ')', 'N');
   END LOOP;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- *****************************************************************************
--
--                          TABLESPACE
--
-- *****************************************************************************
CREATE OR REPLACE PROCEDURE TABLESPACE_H IS
   AX_ID_GRC         NUMBER:=010;
   AX_FREE           NUMBER(12);
   AX_ALOC           NUMBER(12);
   AX_VL_REG         VARCHAR2(200);
   AX_DT_HST         DATE;
   CURSOR TABLESPACE IS
      SELECT TABLESPACE_NAME, SUM(BYTES) TOTAL FROM SYS.DBA_DATA_FILES
         GROUP BY TABLESPACE_NAME;
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Coleta do valor histórico
   SELECT SYSDATE INTO AX_DT_HST FROM DUAL;
   FOR TBS IN TABLESPACE LOOP
      SELECT SUM(BYTES) INTO AX_FREE FROM SYS.DBA_FREE_SPACE
         WHERE TABLESPACE_NAME=TBS.TABLESPACE_NAME;
      IF AX_FREE IS NULL THEN
         AX_FREE:=0;
      END IF;
      SELECT SUM(BYTES) INTO AX_ALOC FROM SYS.DBA_SEGMENTS
         WHERE TABLESPACE_NAME=TBS.TABLESPACE_NAME;
      IF AX_ALOC IS NULL THEN
         AX_ALOC:=0;
      END IF;
      AX_VL_REG:= LPAD(AX_ALOC,12,' ') || ' - ' || LPAD(AX_FREE,12,' ') || ' - ' ||
         LPAD(TBS.TOTAL,12,' ');
      INSERT INTO HISTORICO VALUES
         (AX_ID_GRC, AX_DT_HST, TBS.TABLESPACE_NAME, AX_VL_REG, 'N');
   END LOOP;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/
CREATE OR REPLACE PROCEDURE TABLESPACE_A
   (AX_VL_PRM_001 IN NUMBER, AX_VL_PRM_002 IN NUMBER) IS
   AX_ID_GRC         NUMBER:=011;
   AX_ID_GRC_REF     NUMBER:=010;
   AX_DT_HST_MAX     DATE;
   AX_DT_HST_REF     VARCHAR2(14);
   AX_NR_AUX         NUMBER:=0;
   AX_NR_DIA         NUMBER:=999999;
   AX_NR_DIA_PRV     NUMBER;
   AX_VL_ALC         NUMBER(12);
   AX_VL_DIA         NUMBER(12);
   CURSOR TABLESPACE_ATUAL IS
      SELECT DE_CMP, DT_HST, VL_REG FROM HISTORICO WHERE ID_GRC=AX_ID_GRC_REF
         AND TO_CHAR(DT_HST,'YYYYMMDDHH24MISS') =
            (SELECT MAX(TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')) FROM HISTORICO
               WHERE ID_GRC=AX_ID_GRC_REF);
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
```

```
-- Verificação do valor limite
   SELECT MAX(DT_HST) INTO AX_DT_HST_MAX FROM HISTORICO WHERE ID_GRC=AX_ID_GRC_REF;
   SELECT MAX(TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')) INTO AX_DT_HST_REF FROM HISTORICO
      WHERE ID_GRC=AX_ID_GRC_REF
      AND DT_HST < (AX_DT_HST_MAX - AX_VL_PRM_001);
   IF AX_DT_HST_REF IS NULL THEN
      AX_DT_HST_REF:= TO_CHAR(AX_DT_HST_MAX,'YYYYMMDDHH24MISS');
   END IF;
   WHILE AX_NR_DIA = 999999 LOOP
      IF TO_CHAR(AX_DT_HST_MAX - AX_NR_AUX,'YYYYMMDD') =
         SUBSTR(AX_DT_HST_REF,1,8) THEN
            AX_NR_DIA:= AX_NR_AUX;
      END IF;
      AX_NR_AUX:=AX_NR_AUX + 1;
   END LOOP;
   IF AX_NR_DIA = 0 THEN
      INSERT INTO ALERTA VALUES
         (AX_ID_GRC, SYSDATE, NULL, AX_VL_PRM_001, 'SEM HISTORICO','N');
   ELSE
      FOR TBA IN TABLESPACE_ATUAL LOOP
         SELECT COUNT(*) INTO AX_VL_ALC FROM HISTORICO
            WHERE ID_GRC=AX_ID_GRC_REF AND DE_CMP=TBA.DE_CMP
            AND TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')=AX_DT_HST_REF;
         IF AX_VL_ALC=0 THEN
            AX_VL_ALC:=TO_NUMBER(SUBSTR(TBA.VL_REG,1,12));
         ELSE
            SELECT TO_NUMBER(SUBSTR(VL_REG,1,12)) INTO AX_VL_ALC FROM HISTORICO
               WHERE ID_GRC=AX_ID_GRC_REF AND DE_CMP=TBA.DE_CMP
               AND TO_CHAR(DT_HST,'YYYYMMDDHH24MISS')=AX_DT_HST_REF;
         END IF;
         IF AX_VL_ALC < TO_NUMBER(SUBSTR(TBA.VL_REG,1,12)) THEN
            AX_VL_DIA:= (TO_NUMBER(SUBSTR(TBA.VL_REG,1,12)) - AX_VL_ALC) / AX_NR_DIA;
            AX_NR_DIA_PRV:= TO_NUMBER(SUBSTR(TBA.VL_REG,16,12)) / AX_VL_DIA;
            IF AX_NR_DIA_PRV < AX_VL_PRM_002 THEN
               INSERT INTO ALERTA VALUES
                  (AX_ID_GRC, SYSDATE, TBA.DE_CMP, LPAD(TRUNC(AX_NR_DIA_PRV),12,' '),
                  'MENOS DE ' || AX_VL_PRM_002 || ' DIAS', 'N');
            END IF;
         END IF;
      END LOOP;
   END IF;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ******************************************************************************
--
--                          CRITICAL_OBJECTS
--
-- ******************************************************************************
CREATE OR REPLACE PROCEDURE CRITICAL_OBJECT_A IS
   AX_ID_GRC         NUMBER:=012;
   AX_VL_REG         VARCHAR2(200);
   AX_FRE            NUMBER(12);
   AX_CID            INTEGER;
   CURSOR SEGMENTS IS
      SELECT TABLESPACE_NAME, OWNER, SEGMENT_NAME, SEGMENT_TYPE, NEXT_EXTENT
         FROM SYS.DBA_SEGMENTS
            WHERE SEGMENT_TYPE IN ('TABLE','INDEX')
               ORDER BY 1, 2, 3;
   CURSOR FREE_SPACE IS
      SELECT TABLESPACE_NAME, MAX(BYTES) BYTES
         FROM DBA_FREE_SPACE
            GROUP BY TABLESPACE_NAME;
   TYPE TABTYP IS
      TABLE OF DBA_FREE_SPACE.TABLESPACE_NAME%TYPE
      INDEX BY BINARY_INTEGER;
   TAB TABTYP;
   TYPE FRETYP IS
      TABLE OF DBA_FREE_SPACE.BYTES%TYPE
```

```
      INDEX BY BINARY_INTEGER;
   FRE FRETYP;
   AX_IN_001         NUMBER(5):=0;
   AX_IN_002         NUMBER(5):=0;
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
   FOR FS IN FREE_SPACE LOOP
      AX_IN_001 := AX_IN_001 + 1;
      TAB(AX_IN_001):= FS.TABLESPACE_NAME;
      FRE(AX_IN_001):= FS.BYTES;
   END LOOP;
   FOR SEG IN SEGMENTS LOOP
      AX_FRE:=0;
      FOR AX_IN_002 IN 1..AX_IN_001 LOOP
         IF TAB(AX_IN_002)=SEG.TABLESPACE_NAME THEN
            AX_FRE:= FRE(AX_IN_002);
         END IF;
      END LOOP;
      IF SEG.NEXT_EXTENT > AX_FRE THEN
         AX_VL_REG:= LPAD(SEG.NEXT_EXTENT,12,' ');
         INSERT INTO ALERTA VALUES
            (AX_ID_GRC, SYSDATE, SEG.TABLESPACE_NAME, AX_VL_REG, SEG.OWNER || '.' ||
            SEG.SEGMENT_NAME || ' (' || SEG.SEGMENT_TYPE || ') NEXT > FREE ' ||
            AX_FRE, 'N');
         AX_CID:= DBMS_SQL.OPEN_CURSOR;
         DBMS_SQL.PARSE(AX_CID, 'ALTER ' || SEG.SEGMENT_TYPE || ' "' || SEG.OWNER ||
            '"."' || SEG.SEGMENT_NAME || '" STORAGE (NEXT ' || AX_FRE || ')',
            DBMS_SQL.NATIVE);
         DBMS_SQL.CLOSE_CURSOR(AX_CID);
      END IF;
   END LOOP;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ********************************************************************************
--
--                          FILA PROCESSAMENTO
--
-- ********************************************************************************
CREATE OR REPLACE PROCEDURE FILA_PROCESSAMENTO
   (AX_VL_PRM_001 IN VARCHAR2) IS
   AX_ID_GRC         NUMBER:=045;
   AX_VL_REG         VARCHAR2(200);
   AX_QT_REG         NUMBER(12);
BEGIN
-- Inicio do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
   SELECT COUNT(*) INTO AX_QT_REG FROM XYZ.FILA_QT;
   IF AX_QT_REG > AX_VL_PRM_001 THEN
      AX_VL_REG:= LPAD(AX_QT_REG,12,' ');
      INSERT INTO ALERTA VALUES
         (AX_ID_GRC, SYSDATE, NULL, AX_VL_REG, 'MAIOR QUE ' || AX_VL_PRM_001, 'N');
      UPDATE XYZ.FILA_QT SET ID_INDICE = ID_INDICE + 900000000000
         WHERE ID_INDICE > 0 AND ROWNUM = 1;
   END IF;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/

-- ********************************************************************************
--
--                          ERROS SGBD
--
-- ********************************************************************************
```

```
CREATE OR REPLACE PROCEDURE ERROS_SGBD
   (AX_VL_PRM_001 IN VARCHAR2, AX_VL_PRM_002 IN VARCHAR2) IS
   AX_ID_GRC         NUMBER:=046;
   AX_DT_UEX_INI     DATE;
   AX_VL_REG         VARCHAR2(200);
   AX_DT_MSG         DATE;
   AX_FILE           UTL_FILE.FILE_TYPE;
BEGIN
-- Inicio do processamento
   SELECT DT_UEX_INI INTO AX_DT_UEX_INI FROM GERENCIAMENTO WHERE ID_GRC = AX_ID_GRC;
   UPDATE GERENCIAMENTO SET DT_UEX_INI = SYSDATE WHERE ID_GRC = AX_ID_GRC;
-- Geração dos alertas
   AX_FILE:= UTL_FILE.FOPEN(AX_VL_PRM_001, AX_VL_PRM_002, 'R');
   LOOP
      UTL_FILE.GET_LINE (AX_FILE,AX_VL_REG);
      IF SUBSTR(AX_VL_REG,21,4) = TO_CHAR(SYSDATE,'YYYY') THEN
         AX_DT_MSG:= TO_DATE(SUBSTR(AX_VL_REG,9,2) || SUBSTR(AX_VL_REG,5,3) ||
            SUBSTR(AX_VL_REG,21,4) || SUBSTR(AX_VL_REG,12,8),'DDMONYYYYHH24:MI:SS');
      END IF;
      IF SUBSTR(AX_VL_REG,1,4) = 'ORA-' AND AX_DT_MSG > AX_DT_UEX_INI THEN
         INSERT INTO ALERTA VALUES
            (AX_ID_GRC, SYSDATE, NULL, TO_CHAR(AX_DT_MSG,'YYYY/MM/DD HH24:MI:SS') ||
            ' ' || AX_VL_REG, 'ERRO NO LOG ' || AX_VL_PRM_001 || ' : ' ||
            AX_VL_PRM_002, 'N');
      END IF;
   END LOOP;
UTL_FILE.FCLOSE(AX_FILE);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN;
-- Final do processamento
   UPDATE GERENCIAMENTO SET DT_UEX_FIM = SYSDATE WHERE ID_GRC = AX_ID_GRC;
   COMMIT;
END;
/
```