PROTÓTIPO DE UM SISTEMA DE CONSULTA A BANCO DE DADOS DISTRIBUÍDO HETEROGÊNEO*

Jomi Fred Hübner (jomi@furb.rct-sc.br)

Catia Silene Possamai (rcinfo@netuno.com.br)

Fernanda Souza (fernanda@inf.furb.rct-sc.br)

Geovânio Batista André (gbatista@inf.furb.rct-sc.br)

Marcos Virgilio da Costa (virgilio@inf.furb.rct-sc.br)

Ricardo Rekowsky (ricardoh@inf.furb.rct-sc.br)

Remo Carlo Bortoluzi (remo@fedavi.rct-sc.br)

Resumo

Este trabalho apresenta uma proposta de solução para o problema de recuperação de informações distribuídas em vários SGBDs diferentes em uma corporação. A proposta constituí-se na especificação de um protótipo MiddleWare que torne transparente ao usuário a localização e a interface dos vários SGBDs, além de viabilizar o acesso na rede Internet.

Abstract

This work presents a solution for the problem of retrieving distributed information stored in different data bases. It's proposed a MiddleWare that serves as an interface for the user to access the data bases in a transparent way.

Palavras-chave

Banco de Dados Distribuído; Banco de Dados de três camadas; MiddleWare

1. Introdução

Tanto na Internet como em muitas instituições, as informações estão dispersas em vários setores e armazenadas em vários Sistemas Gerenciadores de Banco de Dados (SGBDs) diferentes. Neste contexto, em muitos casos o usuário não possui uma interface única para acesso aos dados que estão distribuídos nos vários setores de uma empresa mas, ao contrário, tem que utilizar várias aplicações distintas para faze-lo.

Este trabalho apresenta uma proposta de solução para o problema de recuperação de informações distribuídas propondo um protótipo que torne transparente ao usuário a localização e a interface dos vários SGBDs utilizados na corporação, além de viabilizar o acesso na rede Internet.

O texto deste artigo está organizado da seguinte forma: na seções 2, 3 e 4 são apresentados os conceitos teóricos utilizados na implementação do protótipo: BD distribuídos, BD de três camadas e linguagem Java. Na seção 5 é descrita a especificação e implementação do protótipo bem com seus resultados.

---

* Este artigo é resultado do trabalho da implementação da disciplina de Banco de Dados II ministrada no primeiro semestre de 1998 na Universidade Regional de Blumenau para o curso de Bacharel em Ciências da Computação. Portanto, além dos autores enumerados, é merito de todos os alunos da disciplina naquele semestre.

2. Bancos de dados distribuídos

Desde a década de 1960, pesquisadores vem desenvolvendo estudos sobre Sistemas Gerenciadores de Banco de Dados Relacional e Distribuído, no entanto esses sistemas só se tornaram uma realidade no mercado computacional no princípio da década de 1980. Atualmente, a tecnologia de banco de dados distribuído ainda está em constante mudança.

Com o advento de sistemas distribuídos, os usuários passaram a considerar a possibilidade de distribuir os seus dados em um sistema de banco de dados distribuídos que é composto de uma rede de banco de dados locais, armazenados em diversas máquinas, deve "aparecer" para o usuário como um único banco de dados lógico, instalado em uma única máquina. Cada banco de dados local é controlado pelo seu próprio gerenciador de banco de dados local. Assim, os bancos de dados distribuídos oferecem vantagens potenciais a seus usuários, porém surgem muitas questões relacionadas com o gerenciamento de dados. Diferentes tipos de aplicações requerem diferentes filosofias de distribuição de dados [CER95].

De acordo com [DAT91], um sistema de banco de dados distribuídos (BDD) consiste numa coleção de computadores, conectados através de um sistema de rede de comunicações, no qual cada computador tem um SGBD local, mas os computadores "concordaram" em cooperar, de forma que um usuário em qualquer computador pode ter acesso a qualquer dado na rede, exatamente como se o dado estivesse armazenado no próprio computador do usuário. Um sistema de banco de dados distribuídos é, portanto, um sistema de banco de dados virtualmente centralizado, cujos componentes estão armazenados fisicamente em vários bancos de dados "reais".

Segundo [COU91], um sistema de banco de dados distribuídos existe quando um banco de dados integrado logicamente (integração lógica significa que qualquer nó tem acesso potencial a todo o banco de dados) é fisicamente distribuído sobre diferentes nós de computação interligados por uma rede. Idealmente, a distribuição física deve ser transparente aos programas de aplicação. Define-se um nó de computação como um computador localizado numa área de organização com certas facilidades de processamento. Se todos os SGBD locais são idênticos, então se diz que o sistema é **homogêneo**; em caso contrário é **heterogêneo.**

Como principais vantagens dos BDD [CER95] apresenta, entre outros, as seguintes: (1) Tempo de resposta mais rápido, em função da redução do tráfego no acesso a base de dados – as informações ficam mais próximas do local onde serão utilizadas; (2) compartilhamento aumentando através de interfaces padrões entre partições do banco de dados; e (3) melhoria da confiabilidade pois nem todos os SGBDs locais deverão estar funcionando para o sistema como um todo estar (a Internet é um bom exemplo desta flexibilidade gerada pela distribuição).

3. BD de três camadas

Hoje é comum em várias empresas ter grande diversidade de ambientes operacionais e SGBDs. Dentro desse contexto, principalmente quando se trata de Banco de Dados distribuídos, fica difícil recuperar as informações distribuídas em Bancos de Dados de diferentes fornecedores [HAC93].

Existem diversas propostas para o problema de se desenvolver aplicações que necessitem acessar dados residentes em diferentes SGBDs. Uma proposta geral e viável para a integração destes diferentes SGBDs é a utilização de uma camada de software intermediária entre as aplicações cliente e os vários SDBDs (cf. Figura 1).

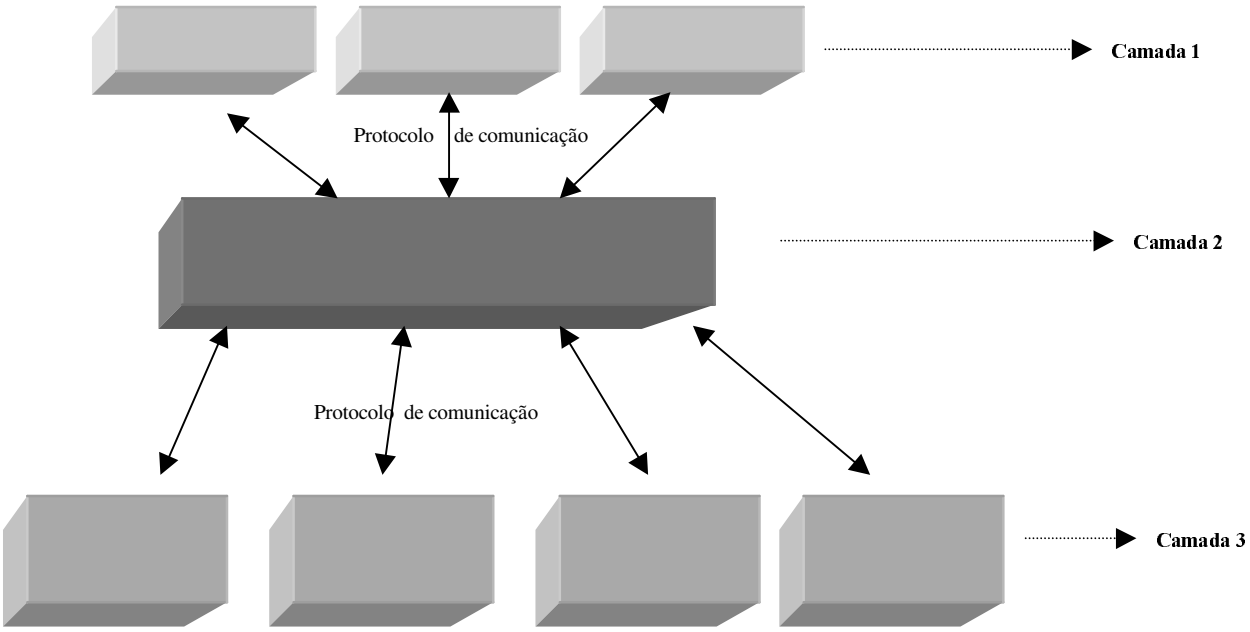

Figura 1: Banco de dados de três camadas

O principal objetivo da camada intermediária, também chamada de *Middleware,* é manter a transparência do tipo de servidor para o cliente. Em outras palavras, os clientes conectam-se ao *Middleware* e submetem-lhe requisições. O Middleware interpreta essas requisições e endereça ao servidor adequado. Assim, o cliente não interage diretamente com o servidor, ficando transparente tanto o tipo de SGBD e sua localização, quanto sua distribuição.

Ao utilizar um BD de duas camadas, que compreenderia o modelo Cliente/Servidor, o usuário teria que acessar os dados interagindo diretamente com o BD disponível no servidor [HAC93]. O BD de três camadas permite que o cliente faça conexão com o *Middleware* como se fosse com um SGBD, e este, por sua vez, verifica a quais BDs o usuário tem acesso, e através da distribuição acessa o computador pertinente ao comando enviado pelo cliente [HAC93]. O próprio *Middleware* recebe a resposta do SGBD e devolve para o cliente o resultado da consulta. Dessa forma, o cliente tem a impressão de que faz acesso com um único BD, independente da multiplicidade de SGBDs existentes nos servidores.

O modelo de três camadas apresenta as seguinte vantagens [SYB98]:

i) Poupam os usuários da complexidade de protocolos e acessos a BD com comandos específicos do SGBD;
ii) Simplificam a integração de fontes de dados heterogêneas e permitem construir sistemas de informações mais facilmente;
iii) Aumenta a habilidade para responder a problemas ou falhas no ambiente distribuído, pois permite uma melhor administração e controle central dos dados;
iv) Flexibilidade em poder ter o *Middleware* num único servidor, pois os clientes se conectam a ele para acessar os SGBDs dos demais servidores. Para disponibilização de informações na Internet, esta característica é necessária.

4. Programação de aplicações distribuídas em Java

Para a implementação do banco de dados em três camadas utilizou-se a linguagem de programação Java que é uma linguagem orientada a objetos (*Object Oriented Language*) originalmente desenvolvida para eletrodomésticos, por isso, ela é simples e portável. Ela nasceu considerando a Internet como seu sistema operacional. "Java herdou várias características de outras linguagens de programação como Objective-C, Smalltalk, Modula 3, entre outras, com o intuito de ser uma linguagem moderna e de fácil manuseio" [CES97].

Ao contrário das ferramentas tradicionais de programação, um programa em Java é compilado para o *bytecode* da Java Virtual Machine, que nada mais é do que uma máquina virtual idealizada pelos criadores da linguagem, e depois executada por uma implementação (em hardware ou software) desta máquina (cf. Figura 2).

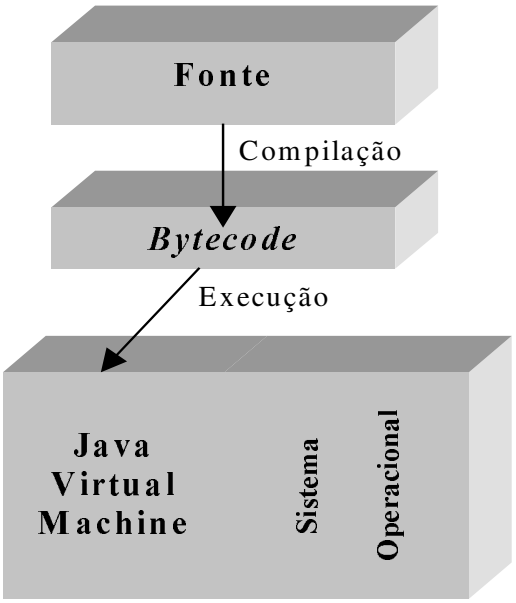

Figura 2: Compilação do fonte para *bytecode* da Máquina Virtual Java (JVM)

O *bytecode* gerado na compilação pode ser transportável para diversas plataformas como Mac/Os, Windows-95/98/NT® , Unix, Linux, por exemplo. Esta característica destaca-se nesse meio por não aparecer em outras linguagens de programação sem que haja uma recompilação do código fonte original com algumas eventuais mudanças.

Além da portabilidade, a linguagem Java também se destaca pela sua confiabilidade. Ela evita erros comuns de programadores gerenciando a memória evitando perda de memória desnecessária através da eliminação do uso de ponteiros, não deixando com que o programador tenha acesso à memória que não pertence ao seu programa.

4.1 Comunicação entre processos remotos: Sockets

Uma das vantagens para utilização de Java para a Internet é que ela oferece facilidades para o desenvolvimento de aplicações com protocolo TCP/IP (sockets, datagramas) utilizado na arquitetura da Internet.

"*Socket* é uma abstração de software usada para representar os "terminais" de uma conexão entre duas máquinas. Para cada conexão há um *socket* para cada máquina, e você pode imaginar que um cabo hipotético está percorrendo entre as duas máquinas, onde no final de cada cabo está conectado um *socket*" [ECK97].

Os *sockets* são usados para implementar conexões confiáveis, bidirecionais e ponto a ponto, com base em *stream*, entre computadores da Internet. Um *socket* pode ser usado para conectar

o sistema de entrada/saída de Java a outros programas que podem residir na máquina local ou em qualquer outra máquina da Internet. Os *sockets* implementam uma conexão persistente altamente confiável entre cliente e o servidor podendo especificar o endereço IP da máquina e sua porta.*

Por exemplo, para oferecer um serviço num determinado computador, utiliza-se um objeto da classe ServerSocket:

```
ServerSocket server = new ServerSocket(8080);
Socket sCliente = server.accept();
PrintStream ps = new PrintStream(sCliente.getOutputStream());
ps.println("Bem vindo");
```

Neste exemplo, a primeira linha cria o serviço na porta 8080 e a segunda espera por uma conexão do cliente. O objeto sCliente viabiliza a comunicação com o cliente.

4.2 Acesso a BD relacional: JDBC

JDBC (Java Data Base Connectivity), como muitas APIs (*Application Program Interface*) em Java, foi projeta para ser simples. Este método provê acesso a SGBDs relacionais através de comandos SQL (*Structured Query Language*). A API utilizada é a mesma para qualquer SGBD o que não implica na necessidade de desenvolver aplicações únicas e exclusivas para um banco de dados específico.

Para ilustrar a utilização da API, considere o seguinte trecho de programa Java como exemplo:

```
Class.forName ("oracle.jdbc.driver.OracleDriver");
Connection conn = DriverManager.getConnection
        ("jdbc:oracle:thin:@mariscal.inf.furb.rct-sc.br:1521:ORAP",
         "scott", "tiger");
Statement stmt = conn.createStatement ();
ResultSet rset = stmt.executeQuery ("select ENAME from EMP");
while (rset.next ())
        System.out.println (rset.getString (1));
}
```

Na primeira linha é feita a carga de um driver para acesso nativo ao BD Oracle; na segunda linha é feita conexão com o BD utilizando o usuário Scott com senha tiger; no terceiro comando, cria-se um ambiente de execução de comandos; no quarto, uma query SQL é submetida ao BD retornando como resultado uma tabela; e, por último, é feito um loop para impressão da primeira coluna da tabela resultante da query.

5. Proposta de acesso a BDs distribuídos

5.1 Funcionamento geral do sistema

A proposta de um protótipo para acesso a BDD e heterogêneos consiste na implementação de um sistema constituído por três camadas (cf. Figura 3):

- **Cliente**: Faz interface a com o usuário. Aceita comandos SQL do usuário e os envia ao Middleware para execução do comando. O Cliente vê o Middleware como um SGBD, e mesmo que os dados recuperados em função do comando enviado pelo cliente sejam de tabelas existentes em vários servidores, isto fica totalmente transparente para o usuário, pois esta resposta é recebida diretamente do Middleware.

---

* Portas não são uma localização física em uma máquina, mas sim uma abstração de software. A idéia é que se for preciso acessar uma porta específica de uma máquina e o serviço associado a essa porta, deve-se requisitar o serviço informando o IP do host e a porta alocada para o serviço [ECK97].

- **Middleware**: serve como intermediário entre os clientes e os diversos SGBDs. Basicamente recebe comandos SQL de um cliente; envia para cada um dos SGBDs conhecidos aonde o usuário tem acesso; espera as respostas e as envia ao cliente. As informações utilizadas pelo Middleware para o seu funcionamento estão armazenadas em um BD chamado de dicionário de distribuição (DD) que armazena os critérios de distribuição (qual SGBD tem que parte dos dados distribuídos), os usuários válidos e suas permissões para acessar aos SGBDs e quais SGBDs e tabelas estão em cada computador.
- O **SGBD**: armazena os dados em tabelas, mantém o relacionamento entre elas permite que sejam acessados esses dados através do MiddleWare, que conecta-se ao SGBD.

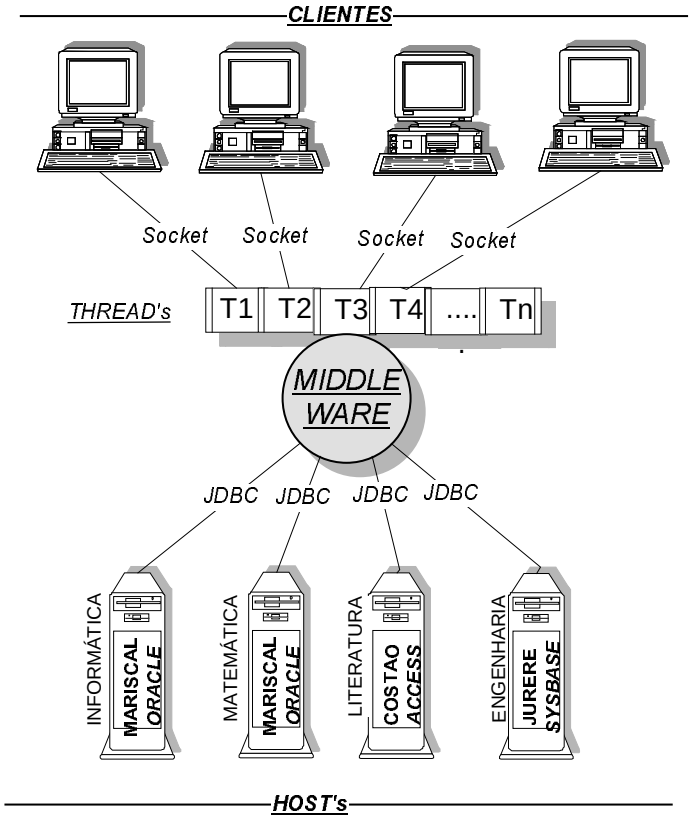

Figura 3: configuração do estudo de caso

5.2 Estudo de Caso

Para estudo e desenvolvimento do protótipo, foi considerado o seguinte caso: uma biblioteca possui cadastro de materiais* distribuídos em bibliotecas setoriais, cada uma com um SGBD local independente. A tabela 1 demonstra um exemplo da distribuição.

O protótipo viabiliza que clientes acessem (via Internet) os BD das bibliotecas de forma transparente, como se esta fosse centralizada e não estivesse distribuída.

* No caso, dos experimentos realizados, foi considerado somente um cadastro de livros e seus assuntos.

Área	SGBD	Máquina (hostname)	Sistema Operacional	Bib
Informática	Oracle	Mariscal	AIX	INF
Literatura	Oracle	Mariscal	AIX	LIT
Biologia	SyBase	Middleware	NT	BIO

Tabela 1: exemplo de distribuição de materiais

O MiddleWare só conecta com os bancos pertinentes ao *select* enviado pelo cliente. Por exemplo, no seguinte select

```
select * from LIVROS where BIB='inf'
```

a query será enviado somente à Mariscal, que é o computador da biblioteca de informática.

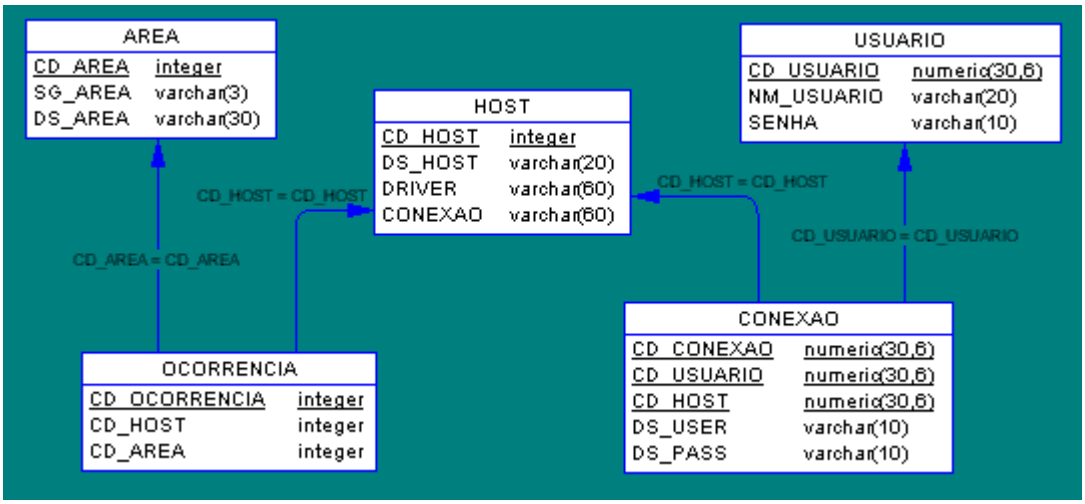

Figura 4: modelo do dicionário de distribuíção

5.3 Modelagem do dicionário de distribuição

A figura 4 mostra o MER (Modelo Entidade Relacionamento) para o DD do Middleware. Segue abaixo a descrição da função de cada entidade e os valores das tabelas no estudo de caso proposto.

Área: Tabela que contém todas as áreas nas quais os livros podem ser agrupados, no caso proposto são as bibliotecas setoriais;

	CD_AREA	SG_AREA	DS_AREA
▶	1	INF	INFORMATICA
	2	LIT	LITERATURA
	3	BIO	BIOLOGIA

Host: Mantém o cadastro de todos os computadores da rede que possuem um SGBD participante do sistema, seu código, descrição, driver de conexão (Oracle, Sybase, ODBC) e o parâmetro de conexão utilizado para conectá-lo.

	CD_HOST	DS_HOST	DRIVER
▶	1	midle	jdbc:odbc:midle
	2	mariscal	oracle.jdbc.driver.OracleDrive

Ocorrência: Relaciona as áreas com os computadores da rede.

	CD_OCORRENCIA	CD_HOST	CD_AREA
▶	1	1	1
	2	2	2
	3	2	3

Usuário: Onde todos os usuários do sistema ficam armazenados, juntamente com sua senha de acesso.

	CD_USUARIO	NM_USUARIO	SENHA
	1	Marcos	****
I	2	Virgillio	*****

Conexão: Relaciona os usuários aos computadores que ele tem acesso, juntamente com seu usuário e senha para que efetue a conexão.

	CD_CONEXAO	CD_USUARIO	CD_HOST	DS_USER	DS_PASS
▶	1	2	2	scott	tiger
	2	1	1	dba	sql
	3	1	2	scott	tiger

5.4 Funcionamento do programa cliente

A comunicação entre o programa cliente e o middleware se dá por meio de conexões para redes TCP/IP, ou seja, sockets. No caso, os dois programas comunicam-se pelo envio de strings segundo especificado na Figura 5.

C ⇒ M: **user** <usuário>	Usuário que deseja fazer conexão
C ⇒ M: **pass** <senha>	Senha do Usuário
M ⇒ C: **ack** | **nack**	*"ack"* se é usuário válido e "n*ack*" se inválido

Se **ack**

C ⇒ M: <comando>	Comando SQL para efetuar consulta
M ⇒ C: <nome>; <nome>, ... M ⇒ C: <tipo>; <tipo>; ...	Nome e tipo das colunas resultantes da consulta
(M ⇒ C: <valor>; <valor>; ...)*	Linhas resultantes da consulta
M ⇒ C: **fim**	A última linha da tabela será *"fim"*

Se **nack**

M ⇒ C: **erro**	Erro de usuário ou senha inválida

Figura 5: Protocolo de Comunicação entre Cliente e Middleware*

Para exemplificar a utilização do protocolo, considere a seguinte comunicação:

C ⇒ M: user marcos
C ⇒ M: pass temp
M ⇒ C: ack
C ⇒ M: select * from livro
M ⇒ C: autor; titulo; local; editora; ano; bib
M ⇒ C: string; string; string; string; integer; string

---

* **Notação**: C é cliente, M é middleware. C ⇒ M: significa C mandando mensagem para M. Argumentos entre <> são não terminais. Argumentos em negrito são terminais (constantes do protocolo). | é o operador de alternativa. * é operador de repetição.

M ⇒ C: COUCEIRO, Luiz Antonio; Sistemas de Gerência de BDD;;;1991;inf
M ⇒ C: DATE, Christian J; Introdução a Sistemas de Banco de Dados;RJ;Campus;1991;inf
M ⇒ C: FONSECA, Rubens; Agosto;RJ;Companhia das Letras;1995;lit
M ⇒ C: fim

5.5 Funcionamento do middleware

O middleware foi implementado utilizando três classes:

- **Classe MWare** (*MWare.class*)**:** a Classe MWare faz a monitoração de uma determinada porta específica do computador, reservada para a conexão com os clientes. A cada conexão de um cliente (através do socket na porta definida) é instanciada uma *thread* (classe Mthread definida abaixo) que passa a manter e gerenciar esta conexão, permitindo então que o *Middleware* mantenha o controle de vários *Clientes* concorrentemente (cf. Figura 3).
- **Classe MThread** (*MThread.class*): nesta classe estão as rotinas de atendimento ao *Cliente* (implementação do protocolo) e de consultas aos SGBD's (consultas de consistência e validação) via JDBC.
- **Classe Cadastro** (*Cadastro.class*): nesta classe estão definidas todas as operações referentes a usuários, bem como suas permissões para o acesso as bibliotecas. Nela serão encontradas opções de inclusão, exclusão, alteração e consulta de usuários, senhas e devidas permissões. Esta classe determina a configuração do sistema distribuído, sendo o único módulo que tem acesso a alterações no Dicionário de Distribuição.

O algoritmo de funcionamento do Middleware (cf. Figura 6) é o seguinte: Estabelecida a conexão, o usuário deve informar seus dados para validação, "usuário" e "senha", e aguardar uma resposta afirmativa ou negativa (Ack ou Nack conforme descrito no protocolo) do *Middleware*. A conexão só permanecerá se o "usuário" e a "senha" enviados pelo *Cliente* estiverem corretos, o que somente vai ocorrer se o primeiro pacote enviado pelo *Cliente* ao *Middleware* puder ser interpretado, para isso o mesmo deve estar dentro da especificação formal do protocolo de comunicação (cf. Figura 5).

Depois de concedida a permissão, o usuário pode então iniciar as consultas ao sistema. Para cada consulta, em forma de comandos SQL, enviada pelo *Cliente*, o *Middleware* realiza algumas consistências baseadas na análise do SQL. Assim, se o *token* "Where" estiver presente na *query*, será necessário identificar a presença de determinadas "áreas" dentre as condições especificadas pelo cliente, para então buscar as ocorrências dos mesmos na tabela "ocorrências", obtendo desta forma, automaticamente os servidores contidos na tabela "hosts" onde finalmente pode verificar se o usuário conectado ao *Middleware* possui permissão para consultar o SGBD que contêm o assunto desejado, restringindo o acesso somente aos bancos correspondentes. Caso a *query* não possua "Where", o username utilizado na conexão servirá como chave para verificação das permissões de acesso, independente do assunto, dependendo apenas da lista de conexões especificada na tabela conexões.

Assim, se o *Cliente* possuir acesso a biblioteca desejada, ele recebe primeiramente duas linhas referente ao cabeçalho da tabela, contendo o nome e o tipo das colunas, e em seguida os dados referentes a consulta, onde a última linha, sinalizando o final da tabela, é a palavra "fim".

```
inicio Middleware
    loop para sempre
        Espera conexão do cliente
        Cria Thread para atende-lo
    fim loop
fim

inicio Thread
    recebe do cliente o username e senha
    se cliente é válido
        loop
            recebe query do cliente
            hosts = todos os hosts que cliente tem acesso
            se tem "where" na query e tem seleção por atributo de distribuição
                hosts = hosts intersecção (todos os relacionados no where)
            fim se
            para todo h de hosts
                envia a query para h
                aguarda resposta
                manda resposta para o cliente
            fim para
        até query = "fim"
    senão
        envia mensagem de erro
fim
```

Figura 6: algoritmo do Middleware

5.6 Exemplos de Consultas

Exemplo 1: um cliente que deseja consultar todos os livros de todas as bibliotecas simplesmente envia o seguinte comando ao Middleware

*Select * from livros*

Esta linha de comando faz uma consulta na tabela de livros de cada servidor onde o usuário possuir permissão, descrito no dicionário de distribuição e pede para que sejam mostrados todos os campos da mesma.

Exemplo 2: consulta a todos os Livros que tenham o Título de ***java*** na biblioteca de informática

*Select * from livros where titulo='java' and bib = 'inf'*

Neste caso, será efetuado uma consulta na tabela de livros do servidor oracle da mariscal (área de informática) se o usuário possuir permissão de acesso a tal servidor.

6. Conclusões

A proposta de sistema para acesso a BD descrita neste artigo apresenta o resultado esperado, como demostrou a implementação e testes realizados. A viabilidade de acesso a SGBDs de fornecedores diferentes foi resultado da opção pela API JDBC, com ela somente foi necessário armazenar no DD quais os drivers e strings de conexão são utilizadas para cada um dos BD do sistema. Outro fato que corroborou para a interoperabilidade do sistema foi a opção por sockets na comunicação entre cliente e Middleware. Com sockets, o programa cliente pode ser implementado em qualquer linguagem que suporte comunicação em redes TCP/IP.

Entretanto algumas limitações foram colocadas neste primeira versão da implementação e podem vir a ser motivação para novos trabalhos:

- Todos os SGBDs devem ter mesmo modelo de dados: no estudo de caso proposto, por exemplo, todos os SGBD tem que ter a tabela de livros e assuntos.

- Distribuição horizontal: cada SGBD tem parte das linhas da tabela distribuída (Livros, no caso). Contudo, em algumas aplicações deseja-se que a distribuição seja feita também por colunas da tabela (algumas colunas ficam em um BD e outras noutro BD).
- Somente consulta: para a implementação de comandos de alteração (insert e update do SQL) a implementação deveria considerar alguns aspectos importantes de integridade, por exemplo: unicidade de chave primária de uma tabela distribuída, integridade referencial (chave estrangeira para uma tabela remota), commit de duas fases, etc.

7. Referências Bibliográficas

[CER95] CERÍCOLA, Vincent O. **Banco de Dados Relacional e Distribuído**. São Paulo : McGraw-Hill, 1995.

[CES97] CESTA, André Augusto. "**Tutorial: A Linguagem de Programação Java**" Características da Linguagem, Agosto de 1997,<http://www.dcc.unicamp.br/~aacesta> (07/05/1998).

[COU91] COUCEIRO, Luiz Antonio C. da Cunha. **Sistemas de Gerência de Banco de Dados Distribuídos**. Brasília : Livros Técnicos e Científicos, 1991.

[DAT91] DATE, Christian J. **Introdução a Sistemas de Banco de Dados**. Rio de Janeiro : Editora Campus, 1991.

[ECK97] ECKEL, Bruce. "**Thinking in Java". Network Programming**, Maio de 1997,< http://www.EckelObjects.com/Eckel>(07/05/1998).

[HAC93] HACKATHORN, Richard D.. **Conectividade de Bancos de Dados Empresariais**. Rio de Janeiro : Infobook, 1993.

[PIR96] PIRES, Paulo de Figueiredo, MATTORO, Marta L. Queirós. **Aspectos de Interoperabilidade na Arquitetura Heterogênea HIMPAR**. (COPPE/UFRJ).

[SYB98] SYBASE. **"Sybase DirectConnect**". **JConnect for JDBC Fact Sheet**, 1998, <http://www.sybase.com/products/entcon/dircon.html> (22/09/1998).