UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

AGRUPAMENTO E ORDENAÇÃO NAS CONSULTAS EM BANCOS DE DADOS DISTRIBUÍDOS ATRAVÉS DE DRIVER JDBC

FRANCISCO ROEDER

BLUMENAU
2010

2010/1-14

FRANCISCO ROEDER

AGRUPAMENTO E ORDENAÇÃO NAS CONSULTAS EM BANCOS DE DADOS DISTRIBUÍDOS ATRAVÉS DE DRIVER JDBC

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Adilson Vahldick - Orientador

BLUMENAU
2010

2010/1-14

AGRUPAMENTO E ORDENAÇÃO NAS CONSULTAS EM BANCOS DE DADOS DISTRIBUÍDOS ATRAVÉS DE DRIVER JDBC

Por

FRANCISCO ROEDER

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Adilson Vahldick , MSc - FURB
Membro:	Prof. Mauro Marcelo Mattos, Dr. Eng. – FURB
Membro:	Prof. Roosevelt dos Santos Jr., Esp. – FURB

Blumenau, 02/07/2010

Dedico este trabalho a meus pais, minha família e amigos que me apoiaram muito e ao meu orientador que acreditou em meu potencial e me auxiliou quando necessário.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À meus pais, que sempre fizeram tudo o que puderam por mim e sempre me apoiaram.

Aos meus amigos, minha família e minha namorada que foram muito compreensíveis.

Aos meus colegas de trabalho que me auxiliaram e meus chefes que permitiram faltas ao longo do desenvolvimento deste.

Ao meu orientador, Adilson Vahldick, por ter aceitado ser meu orientador, por ter sugerido este trabalho, por ter acreditado no meu potencial e por ter me auxiliado sempre que necessário.

A felicidade do homem depende de si mesmo.

Marco Aurélio, Imperador de Roma

RESUMO

A presente monografia descreve alterações realizadas no *driver* JDBC e na Ferramenta de Edição do Esquema Conceitual Global (ECG) de Gonçalves (2007). Este *driver* tem por objetivo efetuar conexões e executar consultas em bancos de dados heterogêneos, localmente ou remotamente distribuídos, através da técnica de fragmentação horizontal de dados. A Ferramenta de Edição de ECG é responsável pela criação de um arquivo XML que contém o ECG. As alterações realizadas no *driver* envolvem o agrupamento e ordenação de consultas em contexto distribuído, visto que em Gonçalves (2007) isto não era possível. A solução para o agrupamento foi implementar um algoritmo baseado em *hash* e para a ordenação a criação de uma classe comparadora. Quanto a Ferramenta de Edição de ECG, foi criada a rotina para gravação do XML e foram feitas melhorias na interface.

Palavras-chave: Banco de dados distribuído. JAVA. JDBC. SGBDD. Driver. Consultas. Ordenação. Agrupamento.

ABSTRACT

The present monograph describes changes performed on the JDBC driver and in Global Conceptual Schema (GCS) Editing Tool of Gonçalves (2007). This driver aims to provide connections and searchs through heterogeneous databases, local or remotely distributed, through data horizontal fragmentation technique. The GCS Editing Tool is responsible for creating a XML file containing GCS. The changes performed trough the driver involve grouping and ordination of querys in distributed context, whereas in Gonçalves (2007) this was not possible. The solution to gathering was to implement an algorithm based on hash and ordination the creation of a comparator class. Regarding the GCS Editing Tool, it was created the routine to write the XML and improvements were made in the interface.

Key-words: Distributed database. JAVA. JDBC. SGBDD. Driver. Querys. Ordination. Grouping.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

BD – Banco de Dados

BDD – Banco de Dados Distribuído

ECG – Esquema Conceitual Global

JDBC – *Java Database Connectivity*

MER – Modelo Entidade Relacionamento

SGBD – Sistema Gerenciador de Banco de Dados

SQL – *Structure Query Language*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A necessidade de integração de informações distintas, armazenadas em locais diferentes, seja fisicamente ou logicamente, é cada vez mais necessária. Conclui-se isso através de Oracle (2007), que cita “conforme os dispositivos inteligentes multiplicam-se, as redes tornam-se poderosas e os volumes de dados se expandem, cresce a necessidade de bancos de dados integrados”. Segundo Özsu e Valduriez (2001, p. 2), é possível alcançar a integração sem centralização e é exatamente isso que a tecnologia de Banco de Dados Distribuídos (BDD) tenta obter. “podemos definir um banco de dados distribuído como uma coleção de vários bancos de dados logicamente inter-relacionados, distribuídos por uma rede de computadores.” (ÖZSU; VALDURIEZ, 2001, p. 5).

A integração de informações armazenadas de formas diferentes pode trazer benefícios, como por exemplo, em uma empresa multinacional onde as sedes de cada país possuem uma base de dados diferenciada. Ou ainda, haver um portal onde se deseja executar uma consulta entre informações de instituições de mesma finalidade, como universidades, polícias, hospitais entre outros. Além disso, em empresas de pequeno porte e com Bancos de Dados (BD) heterogêneos e gratuitos esta integração se torna mais difícil.

Baseando-se no acima exposto, foi dado continuidade ao trabalho acadêmico de Gonçalves (2007) que teve como propósito a construção de uma solução para que aplicações acessem BDD. Esta solução é composta por um *driver Java Database Connectivity* (JDBC) e uma ferramenta para criar o mapeamento entre as bases de dados. Este mapeamento, denominado de Esquema Conceitual Global (ECG) (ÖZSU; VALDURIEZ, 2001), é armazenado no formato *eXtensible Markup Language* (XML) e o *driver* utiliza esse arquivo para executar as consultas.

O *driver* desenvolvido em Gonçalves (2007) concede suporte às consultas de forma global, ou seja, permite que seja executada uma consulta através de várias bases de dados, relacionando as tabelas e colunas das mesmas através do ECG e retornando as informações de forma unificada. Para exemplificar, um caso onde há uma tabela `CLIENTES` em dois bancos de dados distintos, sejam do mesmo fabricante ou não. Através da ferramenta desenvolvida por Gonçalves (2007), é possível no ECG, criar uma relação entre as colunas destas duas tabelas e definir uma nova tabela em nível lógico, chamada `CLIENTES_GLOBAL`, por exemplo. Ao realizar uma consulta desta nova tabela, a partir do *driver*, seriam retornados registros das tabelas `CLIENTES` de ambas as bases de dados. Quando a fragmentação dos dados é feita desta

forma, usando linhas de duas tabelas, trata-se de fragmentação horizontal (ÖZSU; VALDURIEZ, 2001).

No trabalho de Gonçalves (2007), apesar de permitir consultas utilizando as cláusulas de agrupamento e ordenação, estas só eram executadas em cada um dos bancos envolvidos e não sobre o contexto global. O presente trabalho de conclusão implementou essas melhorias no *driver*.

1.1 OBJETIVOS DO TRABALHO

O objetivo primário deste trabalho é possibilitar o agrupamento e ordenação nas consultas em bancos de dados distribuídos através do *driver* JDBC de Gonçalves (2007).

Os objetivos específicos do trabalho são:

a) disponibilizar uma solução para realizar consultas com agrupamento;
b) disponibilizar uma solução para realizar consultas com ordenação;
c) implementar no `driver` as soluções para agrupamento e ordenação;
d) possibilitar o uso do `PreparedStatement`.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos, com a divisão a seguir.

No segundo capítulo é apresentada a fundamentação teórica para auxiliar o entendimento do trabalho assim como conteúdos pesquisados para tornar possível o desenvolvimento do mesmo.

Já o terceiro capítulo apresenta como ocorreu o desenvolvimento do trabalho, explicando o que foi melhorado no *driver* e na ferramenta de edição de ECG e a forma como isto foi feito. Neste capítulo também é apresentada a operacionalidade do desenvolvimento realizado.

Por fim, no quarto e último capítulo, são apresentadas as conclusões, resultados obtidos, as limitações e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo primeiramente é explanado o conceito de Banco de Dados Distribuído (BDD), citando algumas vantagens na utilização deste conceito e apresentando uma das formas para integração de informações. Ainda neste contexto, são apresentadas as definições de agrupamento e ordenação em banco de dados e os fatores relevantes dentro do escopo de um BDD.

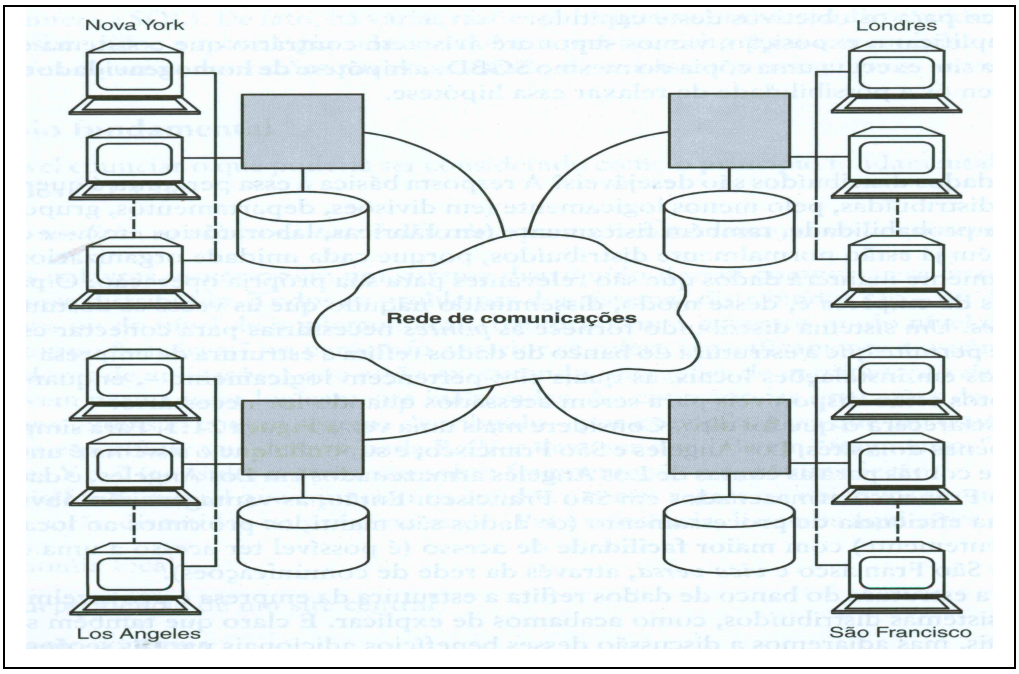
Em seguida, é feita uma abordagem referente à *Application Program Interface* (API) JDBC destacando seus principais recursos e finalidades.

Por fim, são citados alguns trabalhos correlatos, mencionando principalmente o trabalho que será continuado.

2.1 BANCO DE DADOS DISTRIBUÍDO

BDD's são vários bancos de dados fracamente acoplados, interligados através de uma rede de computadores, e que estão logicamente relacionados entre si (ÖZSU; VALDURIEZ, 2001, p. 5) e (SILBERSCHATZ, 1999, p. 589). "Decorre assim que o chamado 'banco de dados distribuído' é na verdade uma espécie de banco de dados virtual, cujas partes componentes estão fisicamente armazenadas em vários bancos de dados 'reais' distintos em vários *sites* distintos" (DATE, 2000, p. 562).

Date (2000, p. 563) define *site* como: "Cada *site* é um sistema de banco de dados por si mesmo. Em outras palavras, cada site tem seus próprios bancos de dados locais 'reais', seus próprios usuários locais, seu próprio Sistema Gerenciador de Banco de Dados (SGBD) local e software de gerenciamento de transações [...]". Na figura 1 é apresentado um exemplo de BDD onde cada *site* encontra-se em uma diferente localidade.



Fonte: Date (2000, p. 563).

Figura 1 - Um sistema de BDD típico

Dentre as vantagens oriundas da utilização de BDD estão a descentralização organizacional e a economia de processamento (ELMASRI e NAVATHE, 2005, p. 580-581). Assim como as empresas possuem divisões lógicas (setores, departamentos e grupos de trabalho), elas possuem também divisões físicas (fábricas, laboratórios e escritórios). Por estas restrições físicas é que os BDDs são desejáveis. Isso se deve a capacidade de armazenar as informações conforme a estrutura da empresa, onde dados locais podem ser mantidos em instalações locais e dados externos podem ser obtidos remotamente (DATE, 2000, p. 564).

"A integração de bancos de dados envolve o processo pelo qual as informações de bancos de dados participantes podem ser integradas conceitualmente para formar uma única definição coesa de um banco de dados múltiplo" (ÖZSU e VALDURIEZ, 2001, p. 570). Isso consiste no processo de projetar um ECG, para que mais de um banco de dados individual possa se tornar um banco de dados múltiplo. A obtenção dos atributos das tabelas de cada BD dentro do ambiente distribuído é identificada quando decompostas as consultas, onde a consulta distribuída é fragmentada entre consultas sobre as relações globais. O ECG que disponibiliza estas relações para que a fragmentação seja possível.

Segundo Silberschatz (1999, p. 589), "nos últimos anos vem aumentando a necessidade de acesso e atualização de dados de um conjunto de bancos de dados preexistentes, com diferentes ambientes de hardware e software e nos esquemas sob os quais esses dados estão armazenados".

2.1.1 Fragmentação

Segundo Özsu e Valduriez (2001, p. 119), “há duas estratégias fundamentais de fragmentação: horizontal e vertical. Além disso, há possibilidade de aninhar fragmentos de forma híbrida.”

A fragmentação horizontal agrupa linhas, para formar subconjuntos de tuplas, onde cada subconjunto de tuplas possui um significado lógico. A partir disso, estes fragmentos podem ser armazenados em diferentes *sites* do sistema distribuído (ELMASRI; NAVATHE, 2005, p. 583-584).

Silberschatz (1999, p. 591) cita que uma relação *r* pode ser particionada em um número de subconjuntos r_1, r_2,.....,r_n. Cada tupla da relação *r* deve pertencer a um desses fragmentos, de modo que a relação possa ser reconstruída se necessário.

Um fragmento horizontal dessa relação *r* pode ser definido como uma seleção (predicado P_i) sobre a relação global conforme quadro 1.

$$r_i = \sigma_{Pi}(r)$$

Fonte: Silberschatz (1999, p. 591).

Quadro 1 - Fragmento horizontal de uma relação

A reconstrução dessa relação à partir desses fragmentos pode ser definida através da expressão algébrica do quadro 2.

$$r = r_1 \cup r_2 \cup \ldots \cup r_n$$

Fonte: Silberschatz (1999, p. 591).

Quadro 2 - Reconstrução de relação à partir de fragmentos

A fragmentação vertical agrupa colunas de uma relação[1], formando subconjuntos de atributos onde a união destes subconjuntos forma a relação completa. Portanto uma relação fragmentada verticalmente pode ser reconstruída da mesma forma que na fragmentação horizontal, através da união dos fragmentos, conforme quadro 2 (SILBERSCHATZ, 1999, p. 592-593).

Um fragmento vertical de uma relação r_i pode ser definido como uma projeção de determinados atributos da relação, conforme quadro 3.

$$r_i = \pi_{Ri}(r)$$

Fonte: Silberschatz (1999, p. 592).

Quadro 3 - Fragmento vertical de uma relação

[1] “Uma organização de dados em uma tabela bidimensional, na qual as linhas (tuplas) representam entidades básicas ou fatos de algum tipo, e coluas (atributos) representam propriedades dessas entidades” (GARCIA-MOLINA, 2001, p. 2).

A fragmentação híbrida, ou mista, é uma fragmentação resultante da combinação de ambas as fragmentações, horizontal e vertical (SILBERSCHATZ, 1999, p. 594).

O grau de fragmentação pode variar entre dois extremos, um onde nenhuma fragmentação é feita e outro, onde os fragmentos seriam compostos por tuplas[2] individuais (na fragmentação horizontal) ou até o ponto de atributos individuais (na fragmentação vertical). O nível de fragmentação afeta o desempenho no processamento de consultas e esse deve ser definido com base nos aplicativos que irão atuar sobre o banco de dados. A consulta para uma relação fragmentada consiste na união destes fragmentos e isso pode ser feito através de junções e reduções (ÖZSU; VALDURIEZ, 2001, p. 116).

2.1.2 Agrupamento

O agrupamento torna-se necessário em consultas quando é preciso aplicar sobre estas funções de agregação como contagem, soma e média. Um exemplo é ter o intuito de obter a média salarial de cada setor ou o número de empregados envolvidos em cada projeto. Outra finalidade das funções de agrupamento é suprimir na consulta, as tuplas que tenham o mesmo valor em algum(ns) de seu(s) atributo(s). A *Structure Query Language* (SQL) possui a cláusula `GROUP BY` para essa finalidade, onde são especificados os atributos de agrupamento, que podem ser listados na cláusula `SELECT` ou não (ELMASRI; NAVATHE, 2005).

As consultas em BDD tornam-se muito mais onerosas do que em bancos de dados centralizados, pois há um número maior de parâmetros que influenciam o desempenho dessas. Consultas distribuídas podem ser fragmentadas e/ou replicadas, elevando os custos de processamento e comunicação. Outro fator é que se forem acessados muitos sites, o tempo de resposta pode se tornar muito alto (ÖZSU; VALDURIEZ, 2001, p. 200).

Ainda segundo Özsu e Valduriez (2001, p. 206), projetos de BDD que contenham operações com eliminação de duplicatas e agrupamento exigem que cada tupla da relação seja comparada com as outras respectivas tuplas e por esse motivo têm complexidade elevada.

[2] Uma linha (registro) de uma tabela (DATE, 2000, p. 97-98).

2.1.3 Ordenação

A SQL torna possível a ordenação das tuplas do resultado de uma consulta, podendo ordenar por apenas um ou vários atributos (colunas), de forma ascendente (que é o padrão da SQL) ou de forma descendente. A ordenação pode ser obtida através da cláusula `ORDER BY` e a especificação da ordenação de forma descendente pode ser feita através da cláusula `DESC`. Para explicitar a ordenação ascendente, pode ser utilizada a cláusula `ASC`. A ordenação pode ser feita em vários níveis, apenas definindo após a cláusula a sequência que a consulta será ordenada. Pode-se tomar como exemplo, a necessidade de listar funcionários, ordenando primeiro pelo setor, depois pelo último nome e finalmente a do primeiro nome (ELMASRI; NAVATHE, 2005, p. 163).

Segundo Özsu e Valduriez (2001, p. 262), a ordenação de junções é um fator importante sobre a otimização de consultas centralizadas, porém em um contexto distribuído é ainda mais importante, devido às junções entre fragmentos poderem aumentar o tempo de comunicação.

2.2 DRIVER JDBC

A API JDBC é uma API Java para acessar virtualmente quaisquer tipos de dados tabulares. Consiste em um conjunto de classes e interfaces escritas na linguagem Java que proporciona uma API padrão para desenvolvedores de aplicações voltadas a bancos de dados que utilizam para esse fim a linguagem Java (WHITE et al., 1999, p. 13-14).

Conforme Ramon (2001, p. 7), “Um programa Java utiliza uma API JDBC única que independe do banco de dados ou *driver* que está sendo utilizado”.

Segundo Reese (2000, p. 25-27), trabalhando com líderes no segmento de banco de dados, a Sun desenvolveu uma API única para acesso a banco de dados. Como parte deste processo mantiveram três principais objetivos:

a) JDBC deve ser uma API em nível de SQL, deve permitir que possam ser criados comandos SQL e executados dentro de qualquer método Java;
b) JDBC deve aproveitar a experiência de APIs de bancos de dados já existentes;
c) JDBC deve ser simples, deve-se poder executar comandos SQL através de poucas

linhas de código.

Um *driver* JDBC é um conjunto de classes que implementam as interfaces JDBC para acesso a um banco de dados em particular. No quadro 4 são apresentadas as principais interfaces JDBC e as suas responsabilidades.

Interface	**Responsabilidade**
`Connection`	Gerenciar a conexão com o banco de dados e permitir criar uma nova consulta através de um `Statement`
`Statement`	Permitir a utilização do método `executeQuery`, que executa a consulta no banco de dados e retornar um `ResultSet` contendo os registros da consulta.
`PreparedStatement`	Mesma da `Statement`, porém deve permitir que a consulta seja feita através de parâmetros, pelos métodos `setString`, `setInt`, entre outros
`ResultSet`	Armazenar o resultado de uma consulta efetuada no BD através de um `Statement`. À partir desta interface é possível percorrer as linhas resultantes da consulta através do método `next`
`ResultSetMetaData`	Fornecer dados sobre o esquema do `ResultSet`, como nome das colunas, quantidade de colunas, tipo e tamanhos

Quadro 4 - Responsabilidades das interfaces JDBC

Quanto a interface `PreparedStatement`, segundo White et al. (1999, p. 326 e 345), a interface herda características da interface `Statement` e difere dela em dois pontos:

a) instâncias de `PreparedStatment` contém um comando SQL que já foi compilado, fazendo com que a consulta já fique “preparada”;
b) o comando SQL contido em um `PreparedStatement` pode ter vários parâmetros de entrada. Esses parâmetros de entrada não são especificados quando o comando é criado, cada um é definido no comando com interrogação e a aplicação é encarregada de definir valores para estes parâmetros antes de executar o comando.

Devido aos objetos do tipo `PreparedStatement` estarem pré-compilados, a execução pode ser mais rápida do que em objetos do tipo `Statement` (WHITE et al., 1999, p. 63).

2.3 DRIVER JDBC PARA CONSULTAS EM BANCOS DE DADOS DISTRIBUÍDOS FRAGMENTADOS HORIZONTALMENTE

Gonçalves (2007), implementou um protótipo de um *driver* JDBC para consultas entre diferentes bases de dados e até bancos de dados e/ou servidores distintos. Para tal, foi criado uma ferramenta para edição do ECG, que armazena as relações entre as tabelas e os atributos de cada base de dados que são retornados nas consultas.

A análise e validação da consulta SQL foram feitas através de uma *Backus-Naur Form* (BNF) criada para suportar comandos SQL compatíveis com a versão SQL ANSI 1999 (SQL3). A construção da BNF e a criação das classes que fazem as análises léxicas, sintáticas e semânticas foram feitas através do Gerador de Analisadores Léxicos e Sintáticos (GALS). O GALS é um ambiente para geração de analisadores léxicos e sintáticos e permite gerar esses analisadores nas linguagens Java, Delphi e C++ (GALS, 2010).

O processo executado pelo *driver* quando feita uma consulta, pode ser resumido da seguinte forma:

1) o sistema registra o *driver*;
2) o *driver* carrega o XML do ECG;
3) o *driver* efetua as conexões com as bases de dados;
4) o sistema envia uma expressão SQL global;
5) o *driver* interpreta a expressão SQL global conforme ECG para consistências;
6) o *driver* monta o comando correspondente e executa em cada uma das bases;
7) o *driver* armazena o resultado de todas as consultas feitas;
8) o *driver* retorna ao sistema a consulta de forma agrupada.

Porém, existiam algumas restrições nestas consultas. Uma delas é que podiam ser utilizadas na SQL cláusulas de agrupamento, ordenação e funções de grupo, porém estas eram executadas apenas em cada uma das bases envolvidas, não considerando como consulta global.

Estas restrições então originaram o desenvolvimento desta monografia, que foi implementada sobre o *driver* de Gonçalves (2007).

2.4 PROCESSAMENTO DE CONSULTAS

Nesta seção são descritos alguns tipos de algoritmos levantados, referentes a agrupamento e de ordenação de tuplas em consultas. Por suas características, estes algoritmos foram desenvolvidos para serem aplicados com programação de baixo nível e em BDs centralizados. Porém, foram fundamentais para o desenvolvimento em programação de alto nível e em BDD.

2.4.1 Algoritmos de agrupamento e agregação

Garcia-Molina et al. (2001, p. 287) descreve um algoritmo para agrupamento e agregação e classifica o mesmo como sendo um “Algoritmo de uma passagem para operações unárias em operações inteiras”. Este algoritmo, de uma passagem, pode ser utilizado somente em situações onde o tamanho da relação não supera o tamanho da memória.

Uma operação de agrupamento pode fornecer vários atributos de agrupamento ou nenhum atributo. Assim como pode conter um ou mais atributos de agregados. Se for criada uma entrada na memória principal para cada grupo, ou seja, para cada valor dos atributos de agrupamento, então poderão ser varridas as tuplas da relação um bloco por vez.

A entrada de um grupo consiste nos valores dos atributos de agrupamento e nos valores acumulados para cada agregação. Os valores acumulados podem ser obtidos por:

a) para um agregado `MIN(a)` ou `MAX(a)`, registrar o menor ou maior valor obtido, respectivamente, do atributo `a`, visto por qualquer tupla do grupo até então. Alterar o valor mínimo ou máximo da agregação, caso seja apropriado, toda vez que uma tupla deste grupo for vista;
b) para qualquer agregação do tipo `COUNT`, adicionar uma unidade para cada tupla do grupo vista;
c) para agregações do tipo `SUM(a)`, adicionar o valor do atributo `a` à soma vista até então;
d) para agregações do tipo `AVG(a)` devem ser mantidas duas totalizações: a soma do número de tuplas do grupo e a soma dos valores de `a` destas mesmas tuplas. Cada uma deve ser calculada como as agregações `COUNT` e `SUM`, respectivamente. Depois

que vistas todas as tuplas do grupo, o valor da agregação será o quociente da soma pela contagem, para obter a média.

Depois de lidas todas as tuplas da relação para o *buffer* de entradas; e estas contribuírem para as agregações de seus grupos, poderá ser produzida a saída gravando uma tupla para cada grupo (GARCIA-MOLINA et al., 2001, p. 287-288).

Segundo Garcia-Molina et al. (2001, p. 298-299), quando a memória disponível for menor que o tamanho da relação, a opção seria um "Algoritmo de duas passagens baseado em classificação". Tendo-se uma grande relação onde o *buffer* para armazenar a relação for maior que a memória disponível poderá ser feito repetidamente:

a) ler todos os blocos possíveis da relação para a memória;
b) classificar esses blocos na memória principal, utilizando um algoritmo de classificação eficiente;
c) gravar a lista classificada na memória sobre a memória disponível. Essa lista classificada será referenciada como uma das sublistas classificadas da relação.

Dessa forma, para se agrupar a relação conforme os atributos da lista de agrupamento devem-se:

1) ler as tuplas da relação `R`, `M` (memória disponível) blocos por vez, classificar cada `M` blocos usando os atributos de agrupamento da lista `L` como chave de classificação. Gravar cada sublista classificada no disco;
2) usar um *buffer* da memória principal para cada sublista e carregar o primeiro bloco de cada sublista em seu *buffer*;
3) encontrar repetidamente o menor valor chave de classificação (atributos de agrupamento) presente entre as primeiras tuplas disponíveis nos *buffers*. Esse valor `v` se torna o próximo grupo para o qual se deve:
   - preparar o cálculo de todos os agregados na lista `L` deste grupo;
   - examinar cada uma das tuplas que tenham a chave de classificação `v` e acumular as agregações necessárias;
   - se um buffer ficar vazio, substituir o mesmo pelo bloco seguinte da mesma sublista.

Dessa forma, quando não houverem mais tuplas com a chave de classificação `v` disponíveis, deve-se fazer a saída de uma tupla com os valores dos atributos de agrupamento de `L` e nos valores associados das agregações calculadas por grupo (GARCIA-MOLINA et al., 2001, p. 301-302).

Garcia-Molina et al. (2001, p. 312-313) descreve ainda um terceiro tipo de algoritmo

classificado como: “Algoritmo de duas passagens baseado em *hash*”.

Supondo que h é uma função de *hash*, e que h tem tuplas completas da relação como seu argumento, ou seja, todos os atributos da relação fazem parte da chave de *hash*. Com estas informações é apresentado o algoritmo do quadro 5, supondo que as tuplas, embora com comprimento variável, nunca sejam grande demais para caber em um novo *buffer*.

```
Inicializar M-1 depósitos usando M-1 buffers vazios;
FOR cada bloco b da relação R DO BEGIN
    ler bloco b para o M-ésimo buffer;
    FOR cada tupla t em b DO BEGIN
        IF o buffer para o depósito h(t) não tiver espaço para t THEN
            BEGIN
            copiar o buffer para o disco;
            inicializar um novo bloco vazio nesse buffer;
            END;
        copiar t para o buffer do depósito h(t);
    END;
END;
FOR cada depósito DO
    IF o buffer para esse depósito não estiver vazio THEN
        gravar o buffer em disco;
```

Fonte: Garcia-Molina et al. (2001, p. 313).

Quadro 5 - Particionando uma relação R em M-1 depósitos

Para executar a operação $\gamma_L(R)$ (onde γ é o operador de agrupamento, *L* a lista de atributos de agrupamento e R a relação), deve-se começar misturando todas as tuplas de R a M-1 depósitos[3] (onde M representa a memória disponível). Porém, para que todas as tuplas de um mesmo grupo fiquem no mesmo depósito, deve ser escolhida uma função *hash* que dependa apenas dos atributos de agrupamento da lista *L*. Após particionado R em depósitos, pode ser utilizado o algoritmo de uma passagem citado anteriormente, para processar um depósito de cada vez (GARCIA-MOLINA et al., 2001, p. 314).

2.4.2 Algoritmo de ordenação

Silberschatz (1999, p. 395), cita que existem dois casos de classificação de consultas: um onde a relação cabe completamente na memória principal e o outro em que a relação é maior que a memória. Para o primeiro podem ser simplesmente utilizadas técnicas-padrão de classificação como o *quicksort*. O segundo caso, onde a relação não cabe na memória é chamado de “classificação externa” e a técnica mais comum utilizada é o algoritmo “*sort-merge* externo”.

[3] Conjunto de blocos de memória para armazenar grupos de registros (GARCIA-MOLINA et al., 2001, p. 184).

Este algoritmo é apresentado a seguir em duas fases:

a) Primeira fase: executar várias classificações temporárias, conforme quadro 6.

b) Segunda fase: fazer *merge* dos arquivos temporários, a fase de *merge* opera conforme quadro 7.

```
i = 0;
repeat
    leia M blocos da relação, ou o resto da relação, aquilo que
        for menor;
    ordene a parte da relação que está na memória;
    escreva os dados ordenados no arquivo temporário Ri;
    i = i + 1;
until o fim da relação
```

Fonte: Silberschatz (1999, p. 395).

Quadro 6 - Primeiro estágio do algoritmo de classificação *sort-merge* externo

```
leia um bloco de cada um dos N arquivos Ri, para uma página de
  buffer na memória;
repeat
    escolha a primeira tupla (na ordem de classificação) entre
       todas as páginas do buffer;
    escreva a tupla no resultado e apague-a da página de buffer;
    if a página de buffer de qualquer temporário Ri está vazia and
        not fim de arquivo(Ri)
        then leia o próximo bloco de Ri, na página de buffer;
until todas as páginas de buffer que estiverem vazias
```

Fonte: Silberschatz (1999, p. 395).

Quadro 7 - Segunda fase do algoritmo de classificação *sort-merge* externo

Tem-se então como resultado da fase de merge a relação classificada (SILBERSCHATZ, 1999, p. 395).

2.4.3 Processamento em contexto distribuído

Segundo Öszu e Valduriez (2001, p. 205), "[..] o objetivo do processamento de consultas em um contexto distribuído é transformar uma consulta de alto nível sobre um banco de dados distribuído, o qual é visto como um único banco de dados pelos usuários, em uma estratégia de execução eficiente e expressa em uma linguagem de baixo nível sobre bancos de dados locais."

Existem quatro camadas principais para mapear uma consulta distribuída em uma sequencia (otimizada) de operações locais, atuando cada uma em um BD local. Essas camadas executam as funções de:

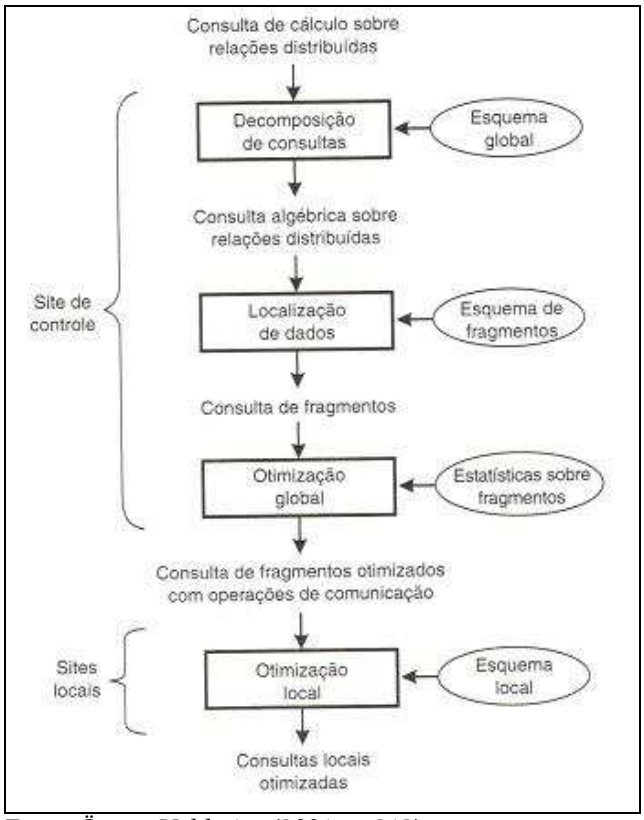
a) decomposição de consultas: decompor a consulta distribuída em uma consulta algébrica sobre relações globais, as informações necessárias para esta

decomposição encontram-se no ECG;

b) localização de dados: localizar os dados da consulta sobre cada um dos fragmentos;
c) otimização de consultas globais: encontrar uma estratégia de execução que esteja próxima da estratégia ótima;
d) otimização de consultas locais: utilizar os algoritmos de sistemas centralizados para otimização local.

Uma visão geral destas funções é apresentada na figura 2.

As três primeiras camadas citadas são executadas pelo *site* central e usam informações em contexto global. Somente a quarta camada é feita pelos *sites* locais (ÖZSU; VALDURIEZ, 2001, p. 211-212).



Fonte: Özsu e Valduriez (2001, p. 212).

Figura 2 - Esquema genérico de camadas para processamento de consultas distribuídas

2.5 TRABALHOS CORRELATOS

Nesta seção são mencionados os trabalhos de Gianisini (2006), Bortolini (2004) e Fink (2000), que de alguma forma possuem relação com este trabalho.

2.5.1 Desenvolvimento de um framework para replicação de dados entre bancos heterogêneos

Gianisini (2006) desenvolveu um *framework* que é capaz de interceptar e replicar as transações de um BD local para um BD remoto. Um fator importante no desenvolvimento desse é que proporciona qualidade de serviço mesmo sem comunicação síncrona e abstrai consideravelmente as bases de dados envolvidas na replicação.

A replicação foi desenvolvida fazendo uma integração através de troca de mensagens, via *Java Message Service* (JMS) e gravando os dados replicados através de *Hibernate*.

As transações globais são controladas através de *Java Transaction* API (JTA), que é uma tecnologia para controle transacional na linguagem Java.

2.5.2 Um protótipo de bancos de dados distribuídos (Caso Expresso São Miguel)

Bortolini (2004) propôs um protótipo de distribuição e replicação de um banco de dados aplicado ao estudo de caso de uma transportadora. Esse protótipo consistiu em um modelo de utilização de um BDD utilizando o SGBD Interbase e uma ferramenta de suporte a replicação, IBReplicator.

No trabalho são apresentadas as técnicas de fragmentação e alocação utilizadas e quais os passos utilizados para replicação dos dados. No trabalho não foi criada uma ferramenta para replicação e sim apenas utilizado o IBReplicator para fazer a replicação. O IBReplicator consiste em um gerente de replicação e um servidor de replicação e não requer uma camada intermediária para conexão com o Interbase. A replicação dos dados foi feita utilizando o modelo assíncrono de transações.

O protótipo foi feito com intuito de aplicá-lo a uma empresa no segmento de transportes na região oeste de Santa Catarina.

2.5.3 Protótipo de Sistema para Acesso a Banco de Dados Distribuídos e Heterogêneos em Redes TCP/IP

Fink (2000) desenvolveu um protótipo de sistema para acesso a BDD utilizando a API JDBC. Nesse trabalho foram criadas duas aplicações: uma para administrar o acesso aos

SGBDs, que foi denominada como Dicionário de Distribuição e outra aplicação em modo console, para permitir a execução dos comandos SQL.

No Dicionário de Distribuição ficam armazenados em uma base de dados criada, os dados necessários para as conexões com os SGBDs. Nesse ainda é possível cadastrar usuários que podem ter acesso ao ambiente distribuído assim como as permissões de acesso. Essas permissões concedem ou não acesso a comandos `SELECT`, `INSERT` ou `UPDATE` e quais tabelas podem ser utilizadas nestes comandos.

Quanto à aplicação responsável pela execução dos comandos SQL, ela o faz apenas replicando comando enviado pelo usuário em cada um dos SGBDs, desde que o usuário possua acesso a base de dados e às tabelas envolvidas no comando.

3 DESENVOLVIMENTO

A apresentação do desenvolvimento deste trabalho foi dividida em duas partes, de acordo com os dois grandes objetivos:

a) implementação do agrupamento e ordenação de consultas no *driver* JDBC;
b) correções e modificações na ferramenta de edição de ECG.

3.1 DRIVER JDBC

Nesta seção são apresentados os requisitos, os diagramas *Unified Modeling Language* (UML), a implementação e exemplos de uso das rotinas de agrupamento e ordenação acrescentadas ao *driver*.

3.1.1 Requisitos

Os requisitos a serem considerados na melhoria do *driver* são:

a) possibilitar o retorno de consultas com agrupamento (`GROUP BY`) sobre o contexto global e não sobre cada *site* (Requisito Funcional – RF);
b) possibilitar o retorno de consultas com as funções de agregação da linguagem SQL (`MIN`, `MAX`, `SUM`, `COUNT` e `AVG`) sobre o contexto global e não sobre cada *site* (RF);
c) possibilitar o retorno de consultas com ordenação de forma ascendente (`ASC`) e descendente (`DESC`) sobre o contexto global e não sobre cada *site* (RF);
d) permitir a utilização da interface `PreparedStatement` da API JDBC para a realização da consultas (RF).

3.1.2 Especificação

Para o desenvolvimento das novas funcionalidades do *driver* foram previamente

especificados os casos de uso, modeladas as classes necessárias e estabelecida a sequência de comunicação entres as classes. Essas etapas foram atendidas com diagramas da UML utilizando a ferramenta Enterprise Architect (EA).

Posteriormente, para realizar os testes dos casos de uso no *driver*, criaram-se esquemas distintos em SGBDs de diferentes fabricantes para simulação, seriam eles MySql, FireBird e Oracle Express Editon (XE). Os Modelos de Entidade e Relacionamento (MER) deram suporte para a modelagem desses esquemas, com a ferramenta DB Designer.

3.1.2.1 Diagrama de Casos de Uso

O diagrama de casos de uso é utilizado pela UML para definir as interações que o usuário ou algum ator específico tem com o software. A figura 3 mostra este diagrama.

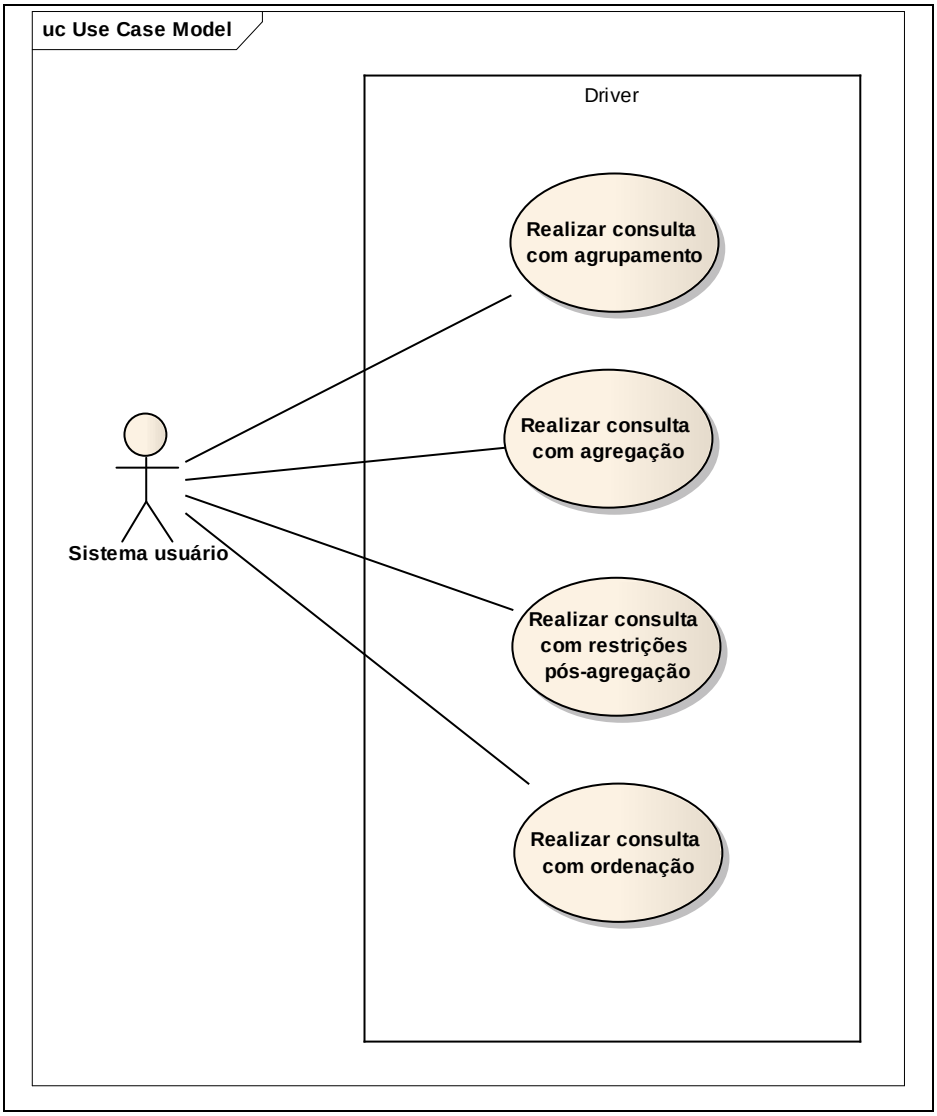


Figura 3 - Diagrama de casos de uso do *driver*

A figura 3 apresenta de forma geral as diferentes formas de utilização do *driver*. O sistema usuário do *driver* pode realizar consultas distribuídas com qualquer combinação de cláusulas `GROUP BY, HAVING` ou `ORDER BY` ou com as funções `SUM`, `MIN`, `MAX`, `COUNT` e `AVG`. A seguir, os quadros 8, 9, 10 e 11 apresentam o detalhamento dos casos de uso.

UC1 – Realizar consulta com agrupamento

Sumário: O sistema usuário realiza uma consulta SQL contendo GROUP BY.

Ator primário: Sistema usuário.

Pré-condições: Nenhuma.

Fluxo principal:

1. O sistema usuário carrega o *driver*.
2. O sistema usuário faz a conexão através de um XML de ECG.
3. O sistema usuário cria um objeto `Statement` através da conexão criada.
4. O sistema usuário cria um comando `SELECT` global, contendo `GROUP BY`.
5. O sistema usuário envia o comando criado através do método `executeQuery`, pelo objeto `Statement`.

Pós-condições: O *driver* retorna um `ResultSet` com a consulta agrupada em contexto global.

Quadro 8 - Descrição do caso de uso `UC1 - Realizar consulta com agrupamento`

UC2 – Realizar consulta com agregação

Sumário: O sistema usuário realiza uma consulta SQL contendo `MIN`, `MAX`, `SUM`, `AVG`, `COUNT`.

Ator primário: Sistema usuário.

Pré-condições: Nenhuma.

Fluxo principal:

1. O sistema usuário carrega o *driver*.
2. O sistema usuário faz a conexão através de um XML de ECG.
3. O sistema usuário cria um objeto `Statement` através da conexão criada.
4. O sistema usuário cria um comando `SELECT` global, contendo `MIN`, `MAX`, `SUM`, `AVG` ou `COUNT`.
5. O sistema usuário envia o comando criado através do método `executeQuery`, pelo objeto `Statement`.

Pós-condições: O *driver* retorna um `ResultSet` com os valores das colunas agregados em contexto global.

Quadro 9 - Descrição do caso de uso `UC2 - Realizar consulta com agregação`

UC3 – Realizar consulta com restrições pós-agregação
Sumário: O sistema usuário realiza uma consulta SQL contendo a cláusula `HAVING`.
Ator primário: Sistema usuário.
Pré-condições: Nenhuma.
Fluxo principal: 1. O sistema usuário carrega o *driver*. 2. O sistema usuário faz a conexão através de um XML de ECG. 3. O sistema usuário cria um objeto `Statement` através da conexão criada. 4. O sistema usuário cria um comando `SELECT` global, contendo a cláusula `HAVING`. 5. O sistema usuário envia o comando criado através do método `executeQuery`, pelo objeto `Statement`.
Pós-condições: O *driver* retorna um `ResultSet` com as restrições das agregações em contexto global.

Quadro 10 - Descrição do caso de uso `UC3 - Realizar consulta com restrições pós-agregação`

UC4 – Realizar consulta com ordenação
Sumário: O sistema usuário realiza uma consulta SQL contendo `ORDER BY`.
Ator primário: Sistema usuário.
Pré-condições: Nenhuma.
Fluxo principal: 1. O sistema usuário carrega o *driver*. 2. O sistema usuário faz a conexão através de um XML de ECG. 3. O sistema usuário cria um objeto `Statement` através da conexão criada. 4. O sistema usuário cria um comando `SELECT` global, contendo `ORDER BY`. 5. O sistema usuário envia o comando criado através do método `executeQuery`, pelo objeto `Statement`.
Pós-condições: O *driver* retorna um `ResultSet` com as colunas da cláusula ordenadas, sobre o contexto global.

Quadro 11 - Descrição do caso de uso `UC4 - Realizar consulta com ordenação`

3.1.2.2 Diagrama de Classes

A figura 4 apresenta o diagrama de classes referentes ao aperfeiçoamento do *driver*.

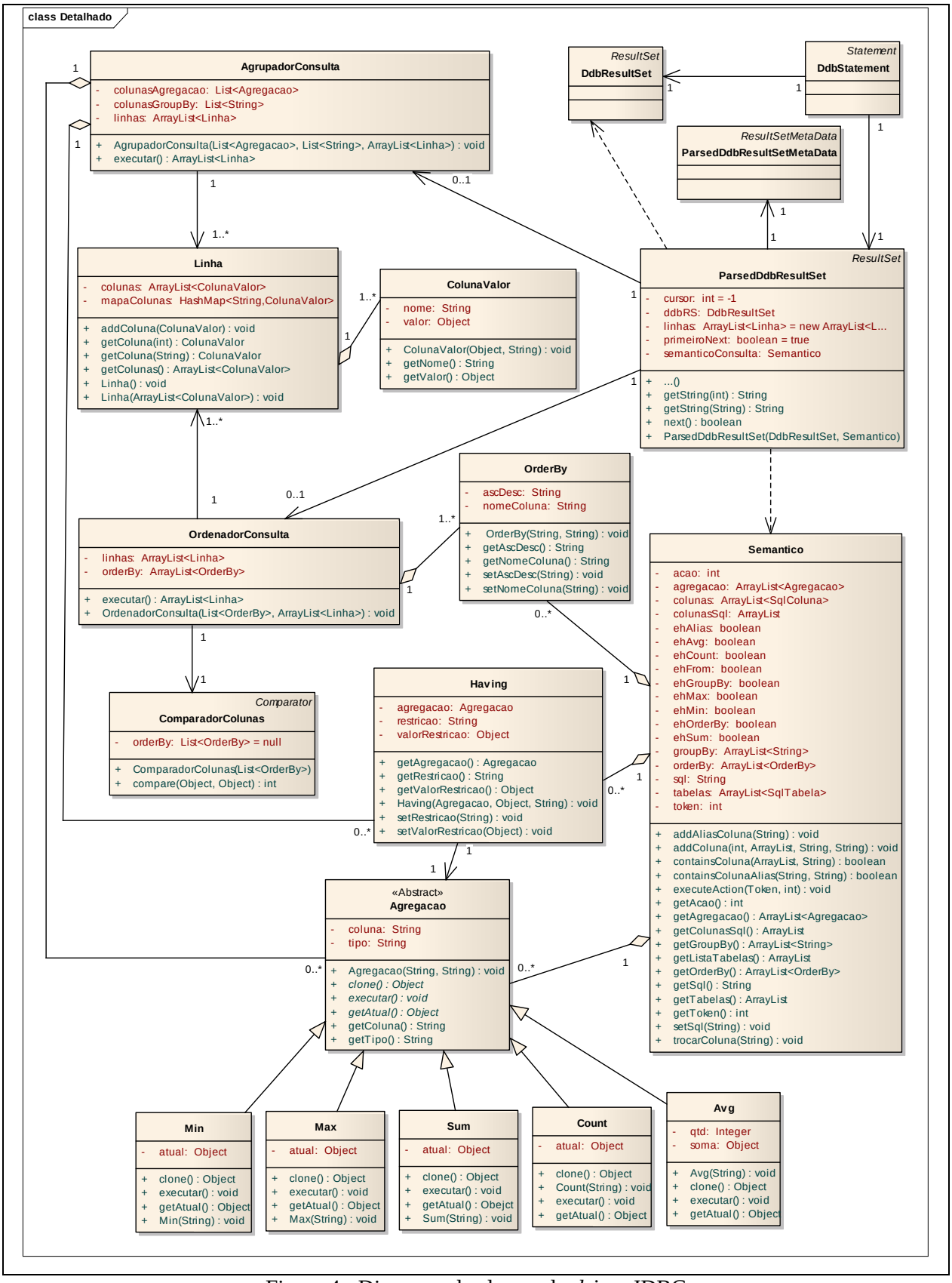


Figura 4 - Diagrama de classes do *driver* JDBC

A seguir, no quadro 12 são listadas as responsabilidades de cada uma dessas classes.

Classe	Responsabilidade
Semantico	Gerada pelo GALS, reescrever o SQL, obter listas de agrupamento, agregação e ordenação
ColunaValor	Armazenar informações de colunas
Linha	Armazenar objetos do tipo ColunaValor
Agregacao	Generalizar objetos específicos de agregação
Min	Armazenar e processar função de agregação MIN
Max	Armazenar e processar função de agregação MAX
Sum	Armazenar e processar função de agregação SUM
Count	Armazenar e processar função de agregação COUNT
Avg	Armazenar e processar função de agregação AVG
Having	Armazenar restrições HAVING
AgrupadorConsulta	Tratar listas de agrupamento, agregação e restrições pós-agregação
OrderBy	Tratar listas de ordenação
ComparadorColunas	Realizar a comparação entre valores de colunas para ordenação
OrdenadorConsulta	Ordenar as linhas da consulta
DdbStatement	Obter o ResultSet através do comando SQL
DdbResultSet	Efetuar a consulta em cada banco de dados
ParsedDdbResultSet	Tratar a consulta efetuada em cada banco de dados
ParsedDdbResultSetMetadata	Controlar tipos de dados da classe ParsedDdbResultSet

Quadro 12 - Responsabilidades das classes envolvidas

A classe Semantico é responsável por obter a listas de atributos de agrupamento, atributos de ordenação e funções de agregação. Estas listas serão repassadas então para a classe ParsedDdbResultSet, que irá efetuar o processamento das linhas oriundas da classe DDBResultSet (já existente no trabalho de Gonçalves (2007)).

Para processar as linhas (registros) oriundas da classe DdbResultSet, foi necessário criar uma estrutura para armazenamento das colunas das mesmas. Essa estrutura é formada pelas classes Linha e ColunaValor, sendo que cada objeto na classe Linha pode armazenar vários objetos da classe ColunaValor e estes por sua vez, armazenam o nome da coluna e o valor da mesma na linha (registro).

Na classe DdbResultSet, se necessário na análise da consulta, são criados objetos das classes AgrupadorConsulta e OrdenadorConsulta. A classe AgrupadorConsulta utiliza para processar o agrupamento e agregações as classes Min, Max, Sum, Count e Avg, que herdam características da classe Agregacao. Ainda nesta classe, são feitas as restrições de agrupamento, utilizando da classe Having para tal. Já a classe OrdenadorConsulta ordena a consulta através da classe OrderBy, com a classe comparadora ComparadorColunas.

3.1.2.3 Diagrama de Sequência

Na figura 5 é apresentado o diagrama de sequência de utilização geral do *driver* JDBC de Gonçalves (2007) com os aperfeiçoamentos implementados.

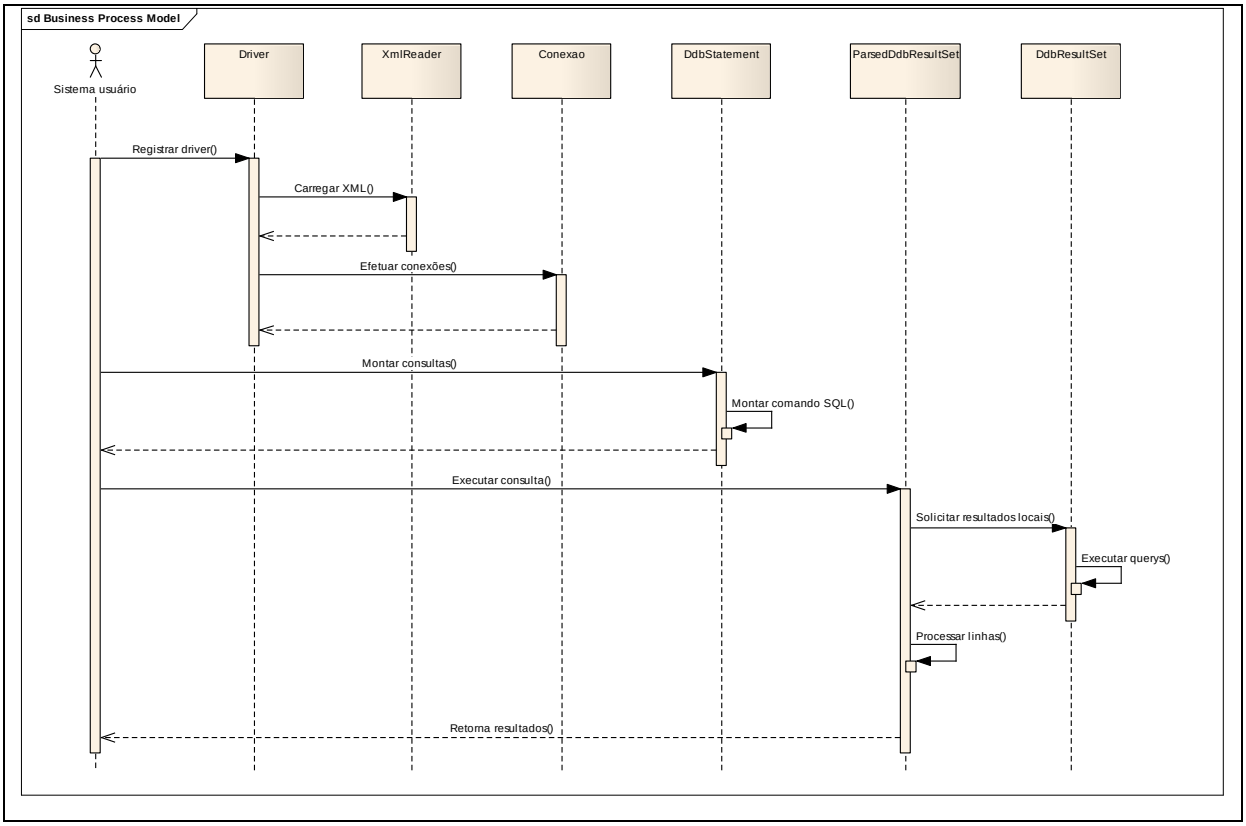


Figura 5 - Diagrama de sequência de utilização geral do *driver* JDBC

O sistema usuário inicialmente registra o *driver* JDBC e obtém uma conexão através do mesmo. Para obter essa conexão, é necessário passar para o *driver* qual o arquivo XML do ECG criado. O *driver* então carrega o XML e a partir deste efetua as conexões com os SGBDs existentes no ECG.

Após as conexões terem sido efetuadas, o sistema usuário cria um `Statement` com a consulta SQL desejada. Neste momento, a classe `DdbStatement` faz a conversão entre a consulta global e as consultas locais. O sistema usuário solicita os resultados através do método JDBC `executeQuery()`. Quando executado este método, o *driver* cria um `DdbResultSet` e um `ParsedDdbResultSet`.

Por fim, o sistema usuário obtém os resultados através de um laço contendo o método JDBC `next()`. Na primeira execução de `next()`, a classe `ParsedDdbResultSet` irá obter todos resultados locais da consulta através da classe `DdbResultSet` e processar o agrupamento e ordenação desses resultados. Esses resultados ficam armazenados em uma

lista, onde a cada chamada do método `next()` pelo sistema usuário, é retornado o próximo resultado desta lista.

3.1.3 Implementação

Os aperfeiçoamentos do *driver* foram desenvolvidos utilizando a IDE NetBeans 6.8. Não foi necessária a inclusão de novas bibliotecas para as alterações realizadas. Sendo que o projeto inicial do *driver* já continha a biblioteca para manipulação de arquivos XML e as bibliotecas para conexão com os bancos de dados.

A primeira tarefa foi a reorganização dos pacotes do projeto do *driver*, deixando as classes separadas em quatro pacotes: `core`, `estrutura`, `gals` e `xml`. Os pacotes também foram renomeados e subdivididos conforme a padronização de pacotes Java, conforme a figura 6.

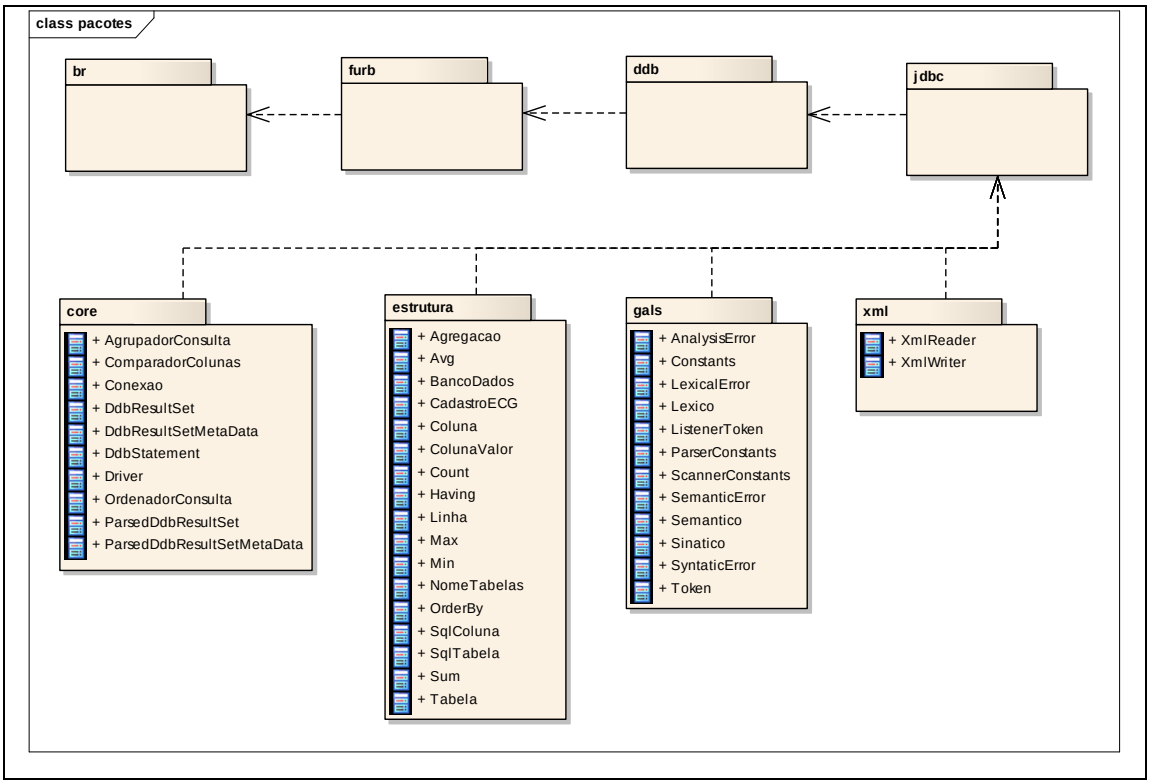


Figura 6 - Diagrama de pacotes do *driver*

O subpacote `jdbc` foi precedido do subpacote `ddb` (*Distributed Data Base*) para indicação do tipo de *driver* JDBC.

O pacote `core` guarda todas as classes referentes ao núcleo do *driver*, estas processam as consultas, como por exemplo: `AgrupadorConsulta` e `OrdenadorConsulta`. Ainda neste pacote ficam também as classes que implementam os métodos das interfaces originais do

JDBC, seriam alguns exemplos: `DdbStatement`, `DdbResultSetMetadata` e `ParsedDdbResultSet`. Além disso, os nomes das classes também foram alterados, substituindo o prefixo `JG` (Jacson Gonçalves, o autor da primeira versão do *driver*) pelo prefixo `ddb`, facilitando o entendimento de que elas tratam o ambiente distribuído.

O pacote `estrutura` guarda as classes de objetos que armazenam as informações utilizadas pelas classes do pacote `core`. Essas informações podem ser: consultas SQL, informações de bancos de dados, nome de tabelas globais, colunas, entre outras. Exemplos dessas classes seriam: `Agregacao`, `OrderBy`, `Min`, `Tabela`, `ColunaValor`, `CadastroECG`, entre outras.

Já o pacote `gals` armazena apenas as classes geradas pelo GALS, seriam algumas delas: `Lexico`, `Semantico`, `Constants` e `SemanticError`.

Por fim, o pacote `xml` guarda apenas duas classes: uma responsável pela leitura de documentos XML com as especificações do ECG e outra pela gravação destes mesmos arquivos XML. Essas classes são: `XmlReader` e `XmlWriter`.

Nas seções seguintes serão demonstrados os aperfeiçoamentos do *driver*.

3.1.3.1 Consultas distribuídas

No quadro 13 é exibido um trecho do método `next` da classe `ParsedDdbResultSet`. Esse método é responsável por retornar a próxima tupla da consulta no contexto distribuído. Cada nova instância da classe `ParsedDdbResultSet` recebe como entrada em seu construtor o *ResultSet* distribuído (`DdbResultSet`) e a instância da classe `Semantico`, que é a responsável pela análise dos comandos `SELECT` obtidos na classe `DdbStatement`.

```
01 public boolean next() throws SQLException {
02   if  (primeiroNext){
03      while (ddbRS.next()){
04         Linha line = new Linha();
05         for (int i = 0; i < ddbRS.getMetaData().getColumnCount(); i++){
06            String nome = (String)ddbRS.getColunasSql().get(i);
07            Object valor = ddbRS.getObject(i+1);
08            ColunaValor col = new ColunaValor(nome,valor);
09            line.addColuna(col);
10         }
11         if (line.getColunas() != null){
12            this.linhas.add(line);
13         }
14      }
15      //Processar agrupamento
16      ArrayList<String> groupBy = semanticoConsulta.getGroupBy();
17      //Processar agregacoes
18      ArrayList<Agregacao> agregacao = semanticoConsulta.getAgregacao();
19      ArrayList<Having> having       = semanticoConsulta.getHaving();
20      if ((groupBy.size() > 0) || (agregacao.size() > 0)){
21         AgrupadorConsulta ac = new AgrupadorConsulta(linhas,groupBy,
22                                              agregacao, having);
23         linhas = ac.executar();
24      }
25      //Processar ordenação
26      ArrayList<OrderBy> listaOrderBy = semanticoConsulta.getOrderBy();
27      if (listaOrderBy.size() > 0){
28         OrdenadorConsulta oc = new OrdenadorConsulta(linhas,
29                                                listaOrderBy);
30         linhas = oc.executar();
31      }
32      primeiroNext = false;
33   }
34   this.cursor++;
35   return this.cursor != linhas.size();
36}
```

Quadro 13 - Método `next` da classe `ParsedDdbResultSet`

A classe `DdbResultSet` já era existente em Gonçalves (2007). O método `next` dessa classe já efetuava a consulta local em cada um dos *sites*. Os passos do método `next` da classe `ParsedDdbResultSet` são:

1) linha 2: verifica se é a primeira execução do método `next` chamado pelo sistema usuário do *driver*;
2) linha 3: inicia um laço para obter as linhas de todos os *sites*, através do `DdbResultSet` origem;
3) linhas 4 a 13 : cria uma nova linha através da estrutura `Linha` e insere esta linha em uma lista de linhas;
4) linha 14: finaliza o laço destinado a obter as linhas das consultas a cada *site*;
5) linha 16: obtenha através do objeto `Semantico`, que foi passado ao construtor, a lista de colunas de agrupamento;

6) linha 18: obtenha também através do objeto `Semantico` a lista de colunas que devem ser agregadas;
7) linha 19: obtenha a lista de restrições `HAVING` da consulta;
8) linhas 20 a 24: verifica se há itens nas listas de agrupamento ou de agregação, caso houver, chama o método `executar` da classe `AgrupadorConsulta` passando as listas obtidas, para que seja processado o agrupamento;
9) linha 26: obtém através do objeto `Semantico`, a lista com as ordenações;
10) linhas 27 a 31: verifica se há itens na lista de ordenação, caso houver, processa a ordenação através do método `executar` da classe `OrdenadorConsulta`;
11) linha 32: seta o atributo `primeiroNext` para falso, para que todo processo descrito acima não seja executado no próximo `next`;
12) linhas 34 a 35: incrementa o cursor para que seja retornada sempre a próxima linha da lista de linhas que foi criada anteriormente e retorna se há ou não próxima linha.

A classe `DdbStatement` já existia no *driver* original de Gonçalves (2007). Essa classe é responsável pela inicialização da consulta e é a classe que o usuário do *driver* utiliza para executar o comando SQL. O método `executeQuery` dessa classe faz as análises léxica, sintática e semântica pelas classes geradas pelo GALS. A BNF utilizada para geração das classes analisadoras pode ser consultada no Apêndice B. Ainda neste método, são feitas as conversões do comando SQL global em vários comandos de contexto local. A criação do `DdbResultSet` e do `ParsedDdbResultSet` é feita no fim do método `executeQuery`, conforme mostrado no quadro 14.

```
public ResultSet executeQuery(String sql) throws SQLException {
...
    DdbResultSet rsDdb = new DdbResultSet(conexoes, sqls,colunasSql,
                                          colunasComFuncao,null);
    return new ParsedDdbResultSet(rsDdb,semantico);
}
```

Quadro 14 - Criação de `ParsedDdbResultSet` na classe `DdbStatement`

3.1.3.2 Agrupamento

Para desenvolvimento do agrupamento, foi necessário alterar a classe `Semantico` do pacote `gals`. Essa classe é responsável pela análise do comando SQL a nível global. No quadro 15 é demonstrado um trecho da classe `Semantico`, responsável pela obtenção das

colunas de agrupamento.

```
public class Semantico implements Constants {
...
    private boolean ehGroupBy = false;
...
    private ArrayList<String>  groupBy = new ArrayList<String>();
...
    public void executeAction(int action,Token token)throws SemanticError{
...
        switch (action) {
...
            case 2: {   //colunas e aliasTabela
...
                if(ehAlias){
                    //Trocar o alias pela coluna
                    trocarColuna(token.getLexeme());
                    ehAlias = false;
                }else
                    //Verificar se a coluna já existe
                    if(!containsColuna(token.getLexeme(), colunas)){
...
                    }
                    if (ehGroupBy){
                        groupBy.add(token.getLexeme());
                    }
...
            }
            case 11:{//group by
                      setSql(getSql() + " group by ");
                      ehGroupBy = true;
                      break;
            }
...
    public ArrayList<String> getGroupBy(){
              return this.groupBy;
    }
...
}
```

Quadro 15 - Tratamento da cláusula `GROUP BY` na classe `Semantico`

Foram criados na classe dois atributos: `ehGroupBy`, que é uma *flag* para indicar se a próxima coluna pertence a cláusula `GROUP BY` e `groupBy`, uma lista que armazena os nomes das colunas que deverão ser agrupadas. Quando é feita a análise semântica, cada *token* da BNF feita por Gonçalves (2007), irá chamar o método `executeAction` passando sua ação. Com essa ação é possível identificar no `switch` a qual cláusula ou trecho do comando SQL se refere o *token*.

A ação 11 indica a cláusula `GROUP BY`, logo, é adicionada essa cláusula ao comando que será feito na consulta local e alterada a *flag* `ehGroupBy`. Já a ação 2 indica que o *token* é referente a uma coluna ou apelido de uma coluna. Porém, como as colunas podem estar em diferentes cláusulas da SQL, faz-se necessário verificar a *flag* criada para identificar se deve ser inserida a coluna na lista de colunas de agrupamento, atributo `groupBy` da classe. Esta lista

com as colunas de agrupamento será retornada através do método público `getGroupBy`, para utilização no método `next` da classe `ParsedDdbResultSet`, conforme à seguir.

O processamento do agrupamento das colunas é feito na classe `AgrupadorConsulta`, conforme linhas 20 a 24 do quadro 13. Esse processamento é feito através do método `executar` da classe, onde um trecho é demonstrado no quadro 16.

```
01 public ArrayList<Linha> executar(){
02    HashMap linhasAgrupadas = new HashMap();
03    for (int i=0; i < linhas.size(); i ++){
04        Linha linha = linhas.get(i);
05        //Limpar valores do groupBy
06        List<Object> groupBy = new ArrayList<Object>();
07        //Verificar se a coluna está no group by
08        for (int j=0; j < colunasGroupBy.size(); j++){
09            String nome2 = colunasGroupBy.get(j);
10            for (int k=0; k < linha.getColunas().size(); k++){
11               String nome1 = linha.getColuna(k+1).getNome();
12               if (nome1.equalsIgnoreCase(nome2)){
13                   groupBy.add(linha.getColuna(k+1).getValor());
14               }
15            }
16        }
17        //Verificar se a linha já está inserida
18        List<Agregacao> agregacoesAtual = (List<Agregacao>)
19                                          linhasAgrupadas.get(groupBy);
20        //Se não estiver na lista ainda insere no HashMap
21        if  (agregacoesAtual == null){
22 ...
23            linhasAgrupadas.put(groupBy,novasColunasAgregacao);
24        }
25 ...
26    }//Fim for das linhas
27 ...
28    Set<ArrayList> chaves = linhasAgrupadas.keySet();
29    ArrayList<Linha> novasLinhas = new ArrayList<Linha>();
30    boolean incluiLinha = true;
31    for (ArrayList lista : chaves){
32        if (lista != null){
33            Linha linha = new Linha();
34            //Adicionar colunas do GroupBy
35            for (int i=0; i < lista.size(); i++){
36                ColunaValor valorColuna = new ColunaValor
37                                    (colunasGroupBy.get(i),lista.get(i));
38                linha.addColuna(valorColuna);
39            }
40 ...
41            if (incluiLinha){
42               novasLinhas.add(linha);
43            }
44        }
45    }
46    return novasLinhas;
47 }
```

Quadro 16 - Agrupamento no método `executar` da classe `AgrupadorConsulta`

O processamento das linhas recebidas no construtor da classe `AgrupadorConsulta`, apresentado no quadro 16 é feito conforme a seguir:

1) linha 2: cria um `HashMap` para armazenar as linhas agrupadas;
2) linha 3: inicia um laço para percorrer todas as linhas da consulta global;
3) linhas 4 a 6: obtém a linha atual do laço e cria uma lista para armazenar os valores dessa linha para as colunas de agrupamento;
4) linha 8: inicia um laço percorrendo a lista de atributos de agrupamento da consulta;
5) linha 9: obtém o nome da próxima coluna que deve ser agrupada, conforme item atual da lista de agrupamento;
6) linha 10: inicia um laço para percorrer todas as colunas da linha atual;
7) linha 11: obtém o nome da coluna atual da linha;
8) linhas 12 a 14: verifica se o nome da coluna de agrupamento é o mesmo da coluna atual da linha, caso seja, insere o valor da coluna na lista `groupBy` criada na linha 6, que será utilizada como chave do `HashMap`;
9) linhas 18 a 24: verifica se já existe no `HashMap` uma chave com a lista `groupBy` obtida, caso não houver, insere no `HashMap` um novo registro tendo como chave esta mesma lista `groupBy`;
10) linhas 28 a 45: percorre o `HashMap` final e cria uma nova lista de linhas, utilizando as chaves e os valores do `HashMap`;
11) linha 46: retorna a nova lista de linhas, que foram agrupadas.

Quando concluído o processo de agrupamento das linhas, estas ficam armazenadas na lista de linhas da classe `ParsedDdbResultSet`. Quando chamado o método `next` pelo sistema usuário, será retornada então a próxima linha da lista que foi armazenada. Esse processo se repete até que chegue ao fim da lista de linhas agrupadas.

3.1.3.3 Agregação

Para o desenvolvimento das funções de agregação nas consultas em contexto distribuído, seguiram-se os mesmos princípios explicados na seção de agrupamento. Porém, como nas agregações são necessárias realizações de cálculos e comparações de valores, foi necessário criar uma estrutura para armazenar essas informações, a classe abstrata `Agregacao`, conforme quadro 17.

```
public abstract class Agregacao{
    private String tipo;
    private String coluna;

    public Agregacao(String tipo, String coluna){
        this.tipo = tipo;
       this.coluna = coluna;
    }

    public String getTipo() {
     return tipo;
    }

    public String getColuna() {
        return coluna;
    }

    @Override
    public String toString(){
        return "" + this.coluna;
    }

    @Override
    public abstract Object clone() throws CloneNotSupportedException;

    public abstract void executar(Linha linha);

    public abstract Object getAtual();
}
```

Quadro 17 - Classe `Agregacao`

Diferente do agrupamento, para cada coluna que utiliza uma função de agregação é necessário guardar também qual o tipo de agregação que deve ser feita. Para isso foram criados os atributos tipo e coluna, que armazenam qual o tipo de agregação e o nome da coluna que será agregada, respectivamente.

Para cada tipo de agregação foi criada uma classe herdeira da classe `Agregacao`: essas classes são `Min`, `Max`, `Sum`, `Count` e `Avg`. Em cada uma dessas classes, os métodos abstratos da classe `Agregacao` foram sobrescritos. A classe `Count` está exemplificada a seguir no quadro 18.

```
public class Count extends Agregacao {
    private Object atual;

    public Count(String col) {
        super("COUNT", col);
        atual = null;
    }

    @Override
    public void executar(Linha linha) {
        if (atual == null){
            atual = (Long)linha.getColuna(super.getColuna()).getValor();
        } else {
            atual = (Long)atual +
                     (Long)linha.getColuna(super.getColuna()).getValor();
        }
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        return new Count(super.getColuna());

    }

    @Override
    public Object getAtual() {
        return atual;
    }
}
```

Quadro 18 - Classe `Count`

Nas classes de cada agregação foi necessário criar um atributo chamado `atual`. Este atributo foi necessário pois o último valor de cada agregação precisa ficar armazenado, para que seja possível efetuar as somas ou comparações. No construtor das classes de agregação é chamado o construtor da classe ancestral (`Agregacao`), passando como parâmetros o tipo da agregação e o nome da coluna que deverá ser agregada.

O método `executar` recebe como entrada uma `Linha` e através dessa incrementa o atributo atual. No caso de `Count`, conforme o algoritmo citado na seção 2.4.1, o atributo `atual` deveria ser apenas incrementado em um. Porém, como a consulta em cada site já é feita com a agregação, é necessário somar os valores obtidos nas linhas ao invés de apenas incrementar um contador. Por exemplo: Se fosse feito `SELECT COUNT(*) FROM X` no *site* n°1 e depois fosse feito `SELECT COUNT(*) FROM Y` no *site* n°2, poderia ser retornado “6” para o *site* n°1 e “10” para o *site* n°2. Sendo assim, se o método `executar` apenas incrementasse o atributo `atual` e não somasse os valores obtidos, o resultado seria “2” ao invés do resultado correto que é “16”.

Além de criar as classes para agregação, foi necessário fazer na classe `Semantico` o mesmo tratamento para criar a lista de agrupamento, citada na seção anterior. Para isso também foi necessário criar uma *flag*, porém agora uma para cada tipo de agregação. No

switch do método executeAction também foi feito tratamento para cada um dos tipos de agregação, a ação da função de agregação COUNT será mostrada como exemplo no quadro 19.

```
public class Semantico implements Constants {
...
    private boolean ehMin = false;
    private boolean ehMax = false;
    private boolean ehSum = false;
    private boolean ehCount = false;
    private boolean ehAvg = false;
...
    private ArrayList<Agregacao> agregacao = new ArrayList<Agregacao>();
...
    public void executeAction(int action,Token token)
                                  throws SemanticError {
...
        case 2: {   //colunas e aliasTabela
...
            if(ehAlias){
                //Trocar o alias pela coluna
                trocarColuna(token.getLexeme());
                ehAlias = false;
             }else{
...
                else if (ehCount){
...
                    else {
                        agregacao.add(new Count(token.getLexeme()));
                        nomeColunasComFuncao.add("COUNT(" +
                                                token.getLexeme()+ ")");
                        ehCount = false;
                    }
                }//count
...
            }//alias
...
        }//case
...
        case 300:{//count
            setSql(getSql() + token.getLexeme());
            ehCount = true;
            break;
        }
...
    }//executeAction
...
}//classe
```

Quadro 19 - Tratamento de COUNT na classe Semantico

A adição de elementos à lista de agregação funciona da mesma forma que para a lista de agrupamentos. A diferença está apenas no tipo de objeto armazenado na lista. Na lista de agrupamentos eram armazenadas apenas itens do tipo String, agora são itens do tipo Agregacao. Os itens adicionados nessa lista de agregações são objetos das classes de agregação criadas: Min, Max, Sum, Count e Avg.

A obtenção dessa lista de agregações na classe ParsedDdbResultSet ocorre da mesma

forma que o agrupamento, conforme visto no quadro 13, no método `next`. O processamento das agregações também ocorre dentro do método `executar` da classe `AgrupadorConsulta`, conforme quadro 20.

```
01 public ArrayList<Linha> executar(){
02 ...
03    for (int i=0; i < linhas.size(); i ++){
04 ...
05        //Verificar se a linha já está inserida
06        List<Agregacao> agregacoesAtual = (List<Agregacao>)
07                          linhasAgrupadas.get(groupBy);
08        //Se não estiver na lista ainda insere no HashMap
09        if  (agregacoesAtual == null){
10            // "agregacao" terao as funcoes agregadoras,
11            //que serao a base para as agregações
12            List<Agregacao> novasColunasAgregacao =
13                                  new ArrayList<Agregacao>();
14            if (colunasAgregacao != null){
15                for (int j =0; j < colunasAgregacao.size(); j++){
16                    try {
17                        novasColunasAgregacao.add((Agregacao)
18                                    colunasAgregacao.get(j).clone());
19                    } catch (CloneNotSupportedException ex) {
20                        Logger.getLogger(AgrupadorConsulta.
21                           class.getName()).log(Level.SEVERE, null, ex);
22                    }
23                }
24            }
25            linhasAgrupadas.put(groupBy,novasColunasAgregacao);
26        }
27        //Obter lista de agregacoes do grupo
28        List<Agregacao> agregacaoAtual = (List<Agregacao>)
29                                        linhasAgrupadas.get(groupBy);
30        //Processa os agregadores no registro atual do HashMap
31        for (int j=0; j < agregacaoAtual.size(); j++){
32            Agregacao ag = agregacaoAtual.get(j);
33            ag.executar(linha);
34        }
35    }//fim for linhas
36 ...
37 }
```

Quadro 20- Agregações no método `executar` da classe `AgrupadorConsulta`

Os passos do método `executar` da classe `AgrupadorConsulta` são:

1) linha 3: percorrer todas as linhas da consulta;
2) linhas 5 a 7: para cada linha, verificar se já existe no `HashMap` um item que tenha como chave a lista das colunas de agrupamento;
3) linhas 9 a 13: criar uma nova lista para armazenar as agregações desse grupo, onde cada item da lista é uma coluna com função de agregação;
4) linhas 14 a 24: inserir na lista de agregações desse grupo um clone de cada agregação, para que se tenham as mesmas colunas e funções de agregação da lista original;

5) linha 25: inserir no HashMap de linhas agrupadas um novo item, tendo como chave a lista de agrupamento e como valor a lista clonada de agregações;
6) linhas 28 a 34: obter a lista de agregações com a chave de agrupamento e nela chamar o método executar de cada uma das agregações.

3.1.3.4 Restrições pós-agregação

As restrições pós-agregação são as restrições feitas através da cláusula SQL HAVING. Para tratar estas restrições também foi necessária a criação de uma estrutura para armazená-las. Esta estrutura foi feita através da criação da classe Having.

```
public class Having {
    private Agregacao agregacao;
    private String restricao;
    private Object valorRestricao;

    public Having(String restricao, String valorRestricao,Agregacao ag){
        this.agregacao      = ag;
        this.restricao      = restricao;
        this.valorRestricao = valorRestricao;
    }
... //Getters e setters
}
```

Quadro 21 - Trecho da classe Having

No atributo restrição fica armazenado o símbolo de restrição que será tratado: "=, <, >, < >, <=, >=". No atributo valorRestricao pode ser armazenado um valor numérico ou um literal, para ser utilizado na comparação. O atributo agregacao armazena qual coluna e função agregadora deverá ser utilizada como base para restrição. Um exemplo de utilização desses três atributos pode ser a restrição: HAVING SUM(VALOR) > 100. Onde SUM(VALOR) estaria no atributo agregacao, ">" estaria no atributo restricao e 100 no atributo valorRestricao.

As restrições HAVING são obtidas na classe Semantico, de forma semelhante aos agrupamentos e agregações. Após obter todas as restrições HAVING do comando SELECT, o tratamento das mesmas é feito na classe AgrupadorConsulta. Esse tratamento é feito ao término do processamento dos agrupamentos e agregações. Como demonstrado anteriormente, no quadro 16, ao final do agrupamento e agregação das colunas, é feita uma iteração sobre o HashMap criado para obter uma nova lista de linhas para retorno. Nesse processo de iteração foi tratada uma condição para validar se a linha inserida atende às restrições HAVING. Para essa validação foi criada o método verificaHaving, que recebe

como entrada um objeto do tipo `Agregacao` e retorna se a linha deve ou não ser inserida no retorno do método `executar` da classe `AgrupadorConsulta`.

```
01 private boolean verificaHaving(Agregacao ag){
02    //So incluir no retorno se atender ao having
03    Boolean incluiLinha = true;
04    for (int j = 0; j < restricoesHaving.size(); j++){
05        incluiLinha = false;
06        //Se for o mesmo tipo de agregacao e coluna, deve restingir
07        if ((restricoesHaving.get(j).getAgregacao().getTipo().
08                        equalsIgnoreCase(ag.getTipo())) &&
09           (restricoesHaving.get(j).getAgregacao().getColuna().
10                     equalsIgnoreCase(ag.getColuna()))){
11            //Verificar restricao
12            if ((ag.getAtual().getClass() == Integer.class) ||
13                (ag.getAtual().getClass() == Long.class) ||
14                (ag.getAtual().getClass() == Float.class) ||
15                (ag.getAtual().getClass() == Double.class) ||
16                (ag.getAtual().getClass() == BigDecimal.class)){
17                Double agregado = Double.parseDouble(ag.getAtual().
18                                                  toString());
19                Double restricao = Double.parseDouble(
20                restricoesHaving.get(j).getValorRestricao().toString());
21                if (restricoesHaving.get(j).getRestricao().equals("=")){
22                    if (agregado.compareTo(restricao) == 0){
23                        incluiLinha = true;
24                    }
25 ...
26                }
27          }
28 ...
29      }
30 ...
31    }
32    //Fim Having
33    return incluiLinha;
34 }
```

Quadro 22 - Trecho do método `verificaHaving` da classe `AgrupadorConsulta`

No quadro 22 pode ser verificado o funcionamento de uma restrição `HAVING` do tipo "=" para um valor numérico conforme a seguir:

1) linha 3: inicializa a *flag* `incluiLinha` como verdadeira;
2) linha 4: inicia laço para percorrer todas as restrições `HAVING`;
3) linhas 5 a 10: caso possua restrição `HAVING`, seta a *flag* `incluiLinha` como falso e verifica se coluna e tipo de agregação da restrição `HAVING` é a mesma da agregação tida como parâmetro;
4) linhas 12 a 16: verifica se a agregação é referente a um valor numérico;
5) linhas 17 a 26: caso o tipo de restrição seja o sinal de igual, verifica se o valor da agregação é igual ao valor da restrição, se for, seta a *flag* `incluiLinha` novamente para verdadeiro;

6) linha 33: retorna a linha deve ou não constar na lista final de linhas, que será retornada através do método `next` da classe `ParsedDddbResultSet`.

3.1.3.5 Ordenação

Para obter o resultado da consulta de forma ordenada, também foi necessária a criação de uma estrutura para armazenar as colunas da cláusula `ORDER BY`. Isso se fez necessário devido ao fato de que além do nome da coluna, é necessário armazenar se a ordenação da mesma deve ser feita de forma ascendente ou descendente. Para isso foi criada a classe `OrderBy` conforme quadro 23.

```
public class OrderBy {
    private String nomeColuna;
    private String ascDesc;

    public OrderBy(String nomeColuna, String ascDesc){
        this.nomeColuna = nomeColuna;
        this.ascDesc = ascDesc;
    }
    /**
    * @return the nomeColuna
    */
    public String getNomeColuna() {
        return nomeColuna;
    }
    /**
    * @return the ascDesc
    */
    public String getAscDesc() {
        return ascDesc;
    }
    /**
    * @param ascDesc the ascDesc to set
    */
    public void setAscDesc(String ascDesc) {
        this.ascDesc = ascDesc;
    }
}
```

Quadro 23 - Classe `OrderBy`

Como as classes anteriores, `OrderBy` foi também inserida em uma lista, na análise do `SELECT` da classe `Semantico`. A lista também é obtida na classe `ParsedDdbResultSet`, porém para o processamento da ordenação foi criada outra classe chamada `OrdenadorConsulta`, apresentada no quadro 24.

```
public class OrdenadorConsulta {
    private ArrayList<Linha> linhas;
    private List<OrderBy> orderBy;

    public OrdenadorConsulta(ArrayList<Linha> linhas,
                             List<OrderBy> orderBy){
        this.linhas             = linhas;
        this.orderBy   = orderBy;
    }

    public ArrayList<Linha> executar(){
        ComparadorColunas comparador = new ComparadorColunas(orderBy);
        Collections.sort(linhas,comparador);
        return linhas;
    }
}
```

Quadro 24 - Classe `OrdenadorConsulta`

Esta classe assim como a classe `AgrupadorConsulta`, recebe uma lista de linhas em seu construtor e também retorna linhas no método `executar`. Esse método apenas utiliza o método `sort` da biblioteca `Collections` para fazer a ordenação da lista de linhas. Porém, para que o método `sort` funcionasse de maneira adequada, seria necessário criar uma classe que implementasse a interface `Comparator`. Foi criada então a classe `ComparadorColunas`. Esta classe recebe em seu construtor a lista de `OrderBy` obtida na análise semântica e através do método `compare` define a ordenação das linhas.

A comparação dos valores das colunas inclusas na cláusula `ORDER BY` teve de ser feita conforme cada classe de objeto desses valores. Ou seja, tiveram de ser comparados `String` com `String`, `Integer` com `Integer` e assim por diante. Isso devido a classe `Object` não ser `Comparable`. No quadro 25 é demonstrado um trecho do método `compare`, desenvolvido na classe `ComparadorColunas`.

Os objetos recebidos como parâmetros do método são duas linhas, que são listas de colunas. Para cada coluna da lista de `OrderBy` passada no construtor, são verificadas cada coluna de uma das linhas se a mesma pertence ao `ORDER BY`. Essa verificação só precisa ser feita em uma das linhas, pois ambas as linhas terão sempre as mesmas colunas e elas também estarão sempre na mesma ordem. Depois, é verificada a classe do valor da coluna, para poder fazer a comparação. É verificado então se a ordenação é ascendente ou descendente e comparados os valores conforme cada um desses tipos.

```
public int compare(Object o1, Object o2) {
    //o1 e o2 são linhas (Listas de colunas)
    Linha linha1 = (Linha) o1;
    Linha linha2 = (Linha) o2;
    int ret = 0;
    //Percorrer a lista de colunas de ordenação
    for (int i=0; i < orderBy.size(); i++){
        //Percorrer cada coluna da linha e verificar
        //se faz parte da ordenação
        for (int j=0; j < linha1.getColunas().size(); j++){
            ColunaValor coluna1 = linha1.getColunas().get(j);
            ColunaValor coluna2 = linha2.getColunas().get(j);
            if ((coluna1.getNome().equalsIgnoreCase(
                        orderBy.get(i).getNomeColuna()))&&(ret == 0)){
                //Testar tipos
                if (coluna1.getValor().getClass() == String.class){
                    String valor1 = (String)coluna1.getValor();
                    String valor2 = (String)coluna2.getValor();
                    if (orderBy.get(i).getAscDesc().
                                    equalsIgnoreCase("desc")){
                        ret = valor2.compareTo(valor1);
                    } else {
                        ret = valor1.compareTo(valor2);
                    }
                }
...
            }
        }
    }
    return ret;
}
```

Quadro 25 - Trecho do método `compare` da classe `ComparadorColunas`

3.2 FERRAMENTA DE EDIÇÃO DE ECG

Nesta seção são apresentadas algumas melhorias e correções feitas na Ferramenta de Edição de ECG, desenvolvida no trabalho de Gonçalves (2007). Estas melhorias visaram apenas melhorar a interface da ferramenta e corrigir erros identificados, sem a inclusão de novos recursos ou funcionalidades. Devido a isso, não foram necessárias especificações para estas alterações. Mais detalhes sobre as funcionalidades e utilização da Ferramenta de Edição de ECG poderão ser encontrados no trabalho de Gonçalves (2007).

3.2.1 Requisitos

Para implementar as melhorias na ferramenta, foram identificados em conjunto com o

orientador alguns requisitos:

a) melhorar a forma de localização de arquivos XML de ECG (RNF);
b) permitir a gravação do XML do ECG (RF);
c) fazer a inclusão e edição do banco local através de nova tela (RNF);
d) corrigir nome do nó raiz da árvore de bancos da aba “Relacionamentos” (RNF);
e) retirar botão “Excluir colunas” da aba “Relacionamentos” (RNF);
f) não exibir as colunas locais já relacionadas, na aba “Relacionamentos” (RNF).

3.2.2 Implementação

O código para gravação do XML do ECG não foi encontrado no projeto original de Gonçalves (2007). Dessa forma, foi necessário criar a classe responsável pela gravação do XML do ECG. Foi criada então a classe `XmlWriter` dentro do pacote `xml` do projeto `DriverJDBC`.

Nessa classe não foi utilizado XML *Schema* para gravação do XML. A gravação foi feita utilizando o padrão UTF-8, através da biblioteca `jdom`.

```
01 public void gravarXML(Document doc, String caminho) throws Exception{
02    StringWriter fos = new StringWriter();
03    OutputFormat of = new OutputFormat("XML","UTF-8",true);
04    XMLSerializer serializer = new XMLSerializer(fos,of);
05    serializer.asDOMSerializer();
06    serializer.serialize( doc.getDocumentElement() );
07    FileWriter out = new FileWriter(caminho);
08    out.write(fos.toString());
09    out.close();
}
```

Quadro 26- Método `gravarXML` da classe `XmlWriter`

O quadro 26 apresenta o código do método `gravarXML`, onde:

1) linhas 2 a 4: cria um serializador XML com o formato de saída no padrão UTF-8;
2) linhas 5 a 6: serializa o documento o objeto `Document` recebido como parâmetro;
3) linhas 7 a 9: grava o documento serializado no caminho recebido como parâmetro.

Além dos métodos `gravarXml` e `gravarXmlBanco`, foi necessária a criação de mais dois métodos: um para criar os elementos das tabelas globais e outro para criar os elementos dos relacionamentos. São estes: `gravarXmlTabelasGlobais` e `gravarXmlRelacionamentos`, respectivamente. No quadro 27 é demonstrado um destes métodos.

```
public Element gravarXmlTabelasGlobais(CadastroECG cad, Document doc){
    //Gravar ECG dos bancos
    //Gravar tabelas e colunas globais
    Element tabelasGlobais = doc.createElement("tabelasGlobais");
    Iterator it = cad.getTabelasGlobal().keySet().iterator();
    while (it.hasNext()){
        @SuppressWarnings("element-type-mismatch")
        String tabela = (String)it.next();
        Element tg = doc.createElement("tabelaGlobal");

        Element nomeTabela = doc.createElement("nome");
        nomeTabela.appendChild(doc.createTextNode(tabela));
        tg.appendChild(nomeTabela);

        Element cols = doc.createElement("colunas");
        //nomeTabela.appendChild(doc.createTextNode(""));

        ArrayList colunas = cad.getTabelasGlobal().get(tabela);
        for (int i=0;i < colunas.size();i++){
            Element coluna = doc.createElement("coluna_" + i);
            String nomeColuna = (String)colunas.get(i);
            coluna.appendChild(doc.createTextNode(nomeColuna));
            cols.appendChild(coluna);
        }
        tg.appendChild(cols);

        tabelasGlobais.appendChild(tg);
    }
    return tabelasGlobais;
}
```

Quadro 27- Método `gravarXmlTabelasGlobais` da classe `XmlWriter`

As demais alterações feitas na ferramenta foram suprimidas nesta seção por se tratar de alterações apenas de interface. Na seção seguinte estas são apresentadas através de telas.

3.2.3 Operacionalidade da implementação

Inicialmente foi alterado o *layout* de tela da primeira aba, `Bancos de Dados`, na Ferramenta de Edição de ECG. Anteriormente a tela estava confusa, pois os campos de cadastro de cada banco de dados ficavam acima da grade e estes campos não eram atualizados conforme banco selecionado na grade. A figura 7 demonstra a tela da versão anterior.

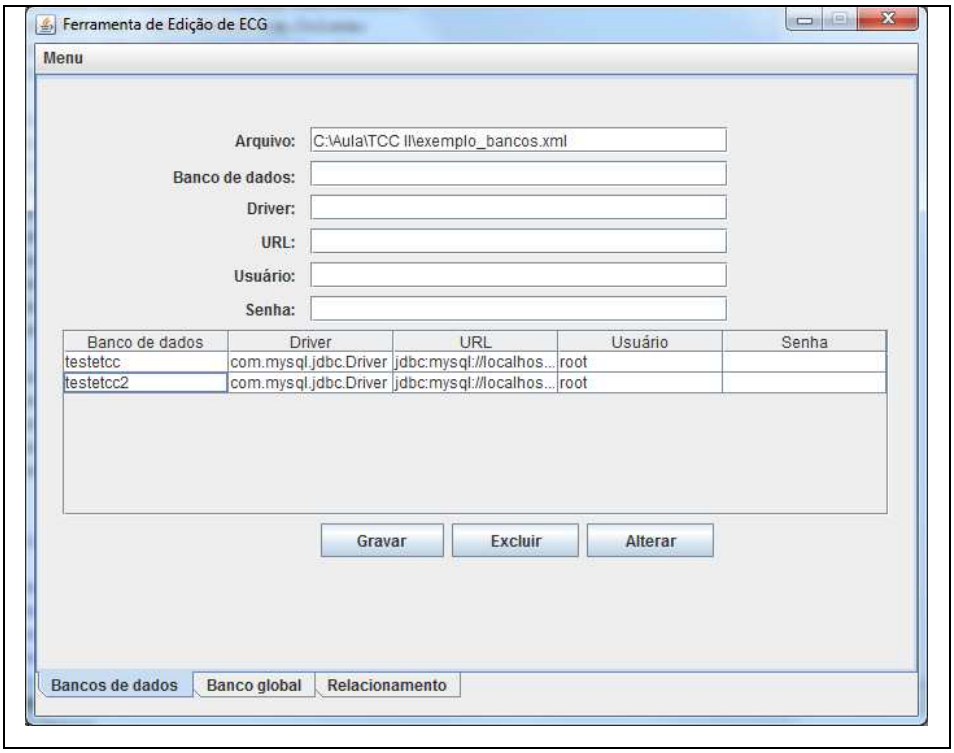


Figura 7 - Tela antiga do cadastro de BDs

Foi alterada esta tela, para apresentar somente a grade de bancos de dados incluídos no ECG, e trazendo os campos de cadastro somente quando incluído ou editado algum banco. A figura 8 mostra como ficou a tela sem nenhum ECG criado ou aberto.

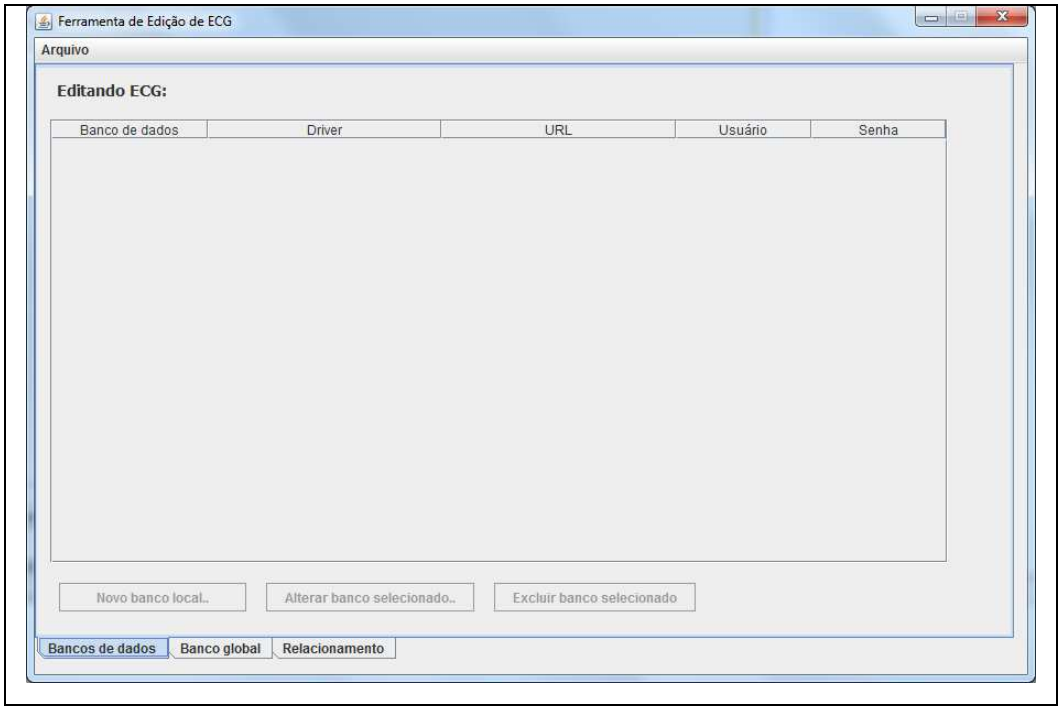


Figura 8 - Tela do cadastro de BDs nova

Conforme a figura 8, O nome do arquivo do ECG que está sendo editado deixou de estar em uma caixa de texto e passou a estar na parte superior, após o *label* “Editando ECG”.

Também foram alterados os textos dos botões inferiores e foram tratados para que só ficassem habilitados quando permitido, conforme figura 8. O menu superior também teve seu nome alterado para “Arquivo” e foram criados atalhos para as opções do menu “Arquivo”.

Agora, para criar um novo ECG deve-se clicar em “Arquivo” e depois “Novo ECG”, como mostra a figura 9.

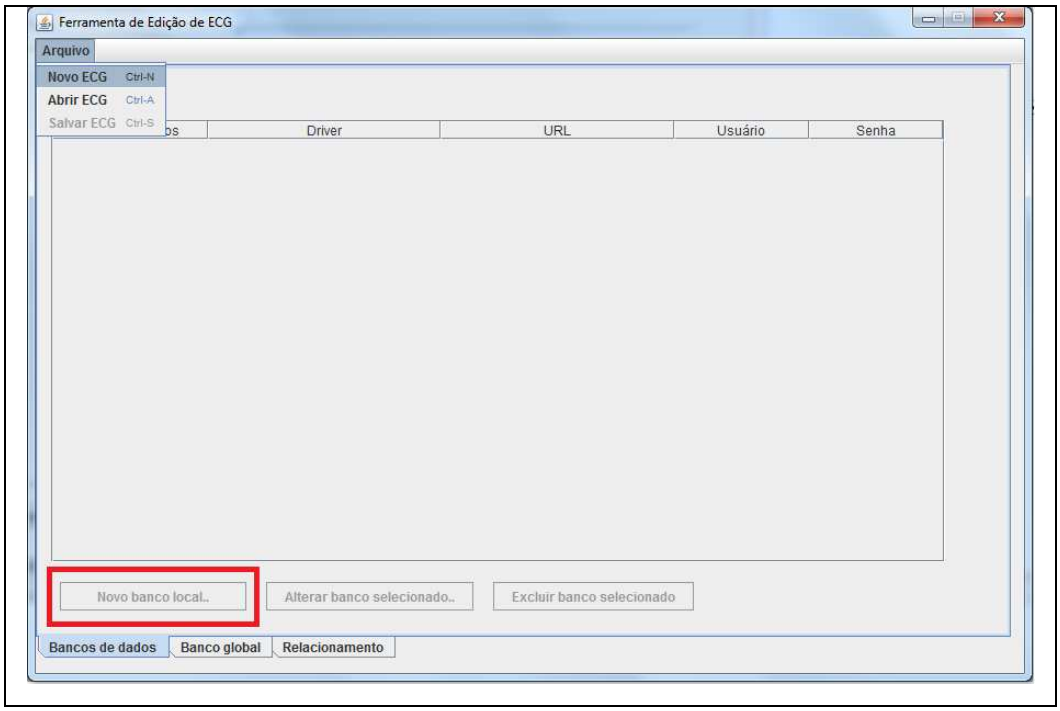


Figura 9 - Criação de ECG na Ferramenta de Edição de ECG

No nome do arquivo aberto irá aparecer “Novo ECG” e o botão “Novo banco local” ficará habilitado (destacado na figura 9). Quando clicado neste botão é que serão apresentados os campos do cadastro do novo BD local, conforme figura 10.

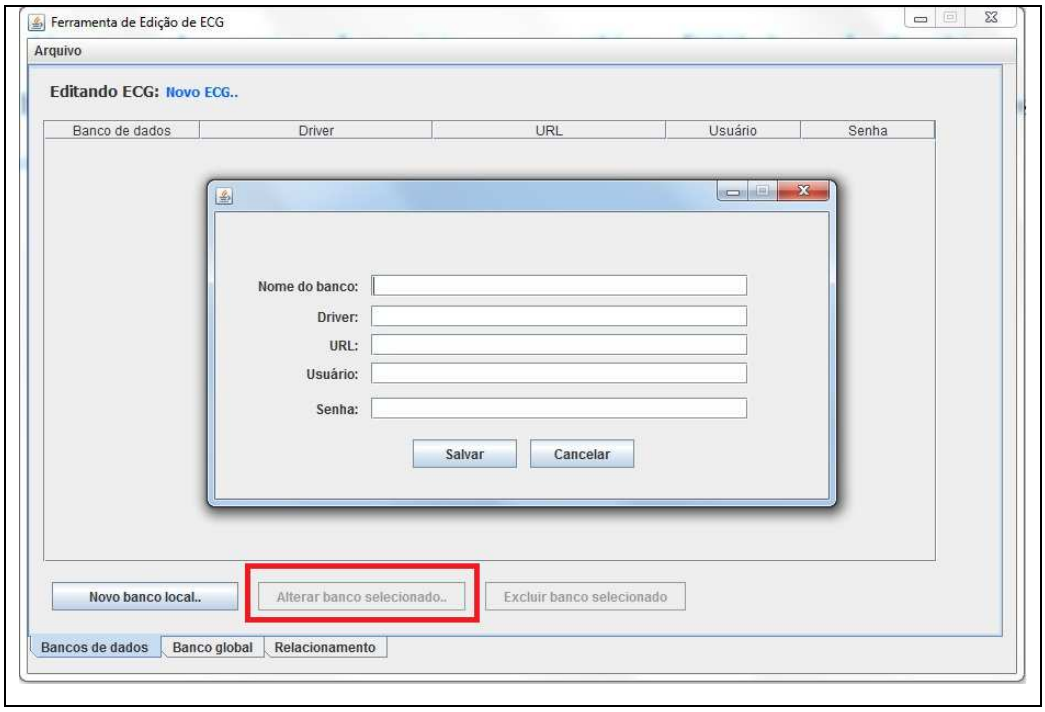


Figura 10 - Tela de cadastro de novo BD

Após salvo o novo banco, o mesmo será listado na grade e poderá ser editado através do botão “Alterar banco selecionado” (destacado na figura 10). Este botão irá abrir a mesma tela, porém com os campos já preenchidos referentes ao banco selecionado na grade.

Caso exista ao menos um banco incluído é possível salvar o ECG no arquivo XML, através do menu “Arquivo” e “Salvar ECG”, será aberta uma caixa para salvar o arquivo, conforme figura 11.

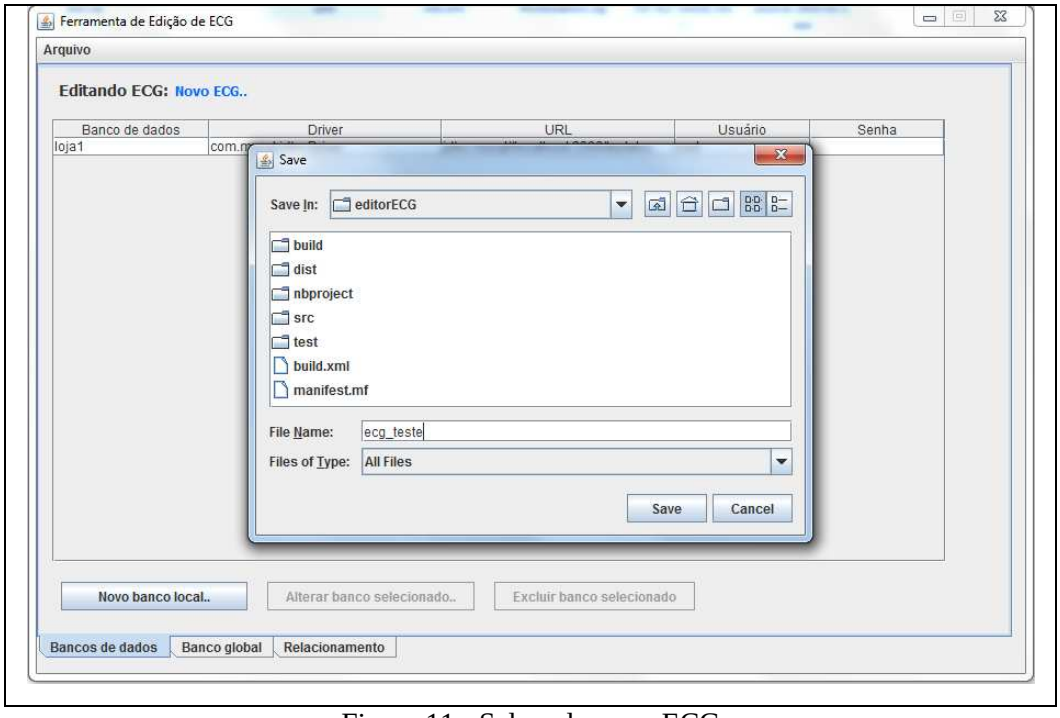


Figura 11 - Salvando novo ECG

Se for salvo um ECG que está sendo editado e não um novo, não será aberta a tela da figura 11, a aplicação apenas irá sobrescrever o arquivo. Após salvo o ECG em arquivo, a tela ficará conforme figura a 12.

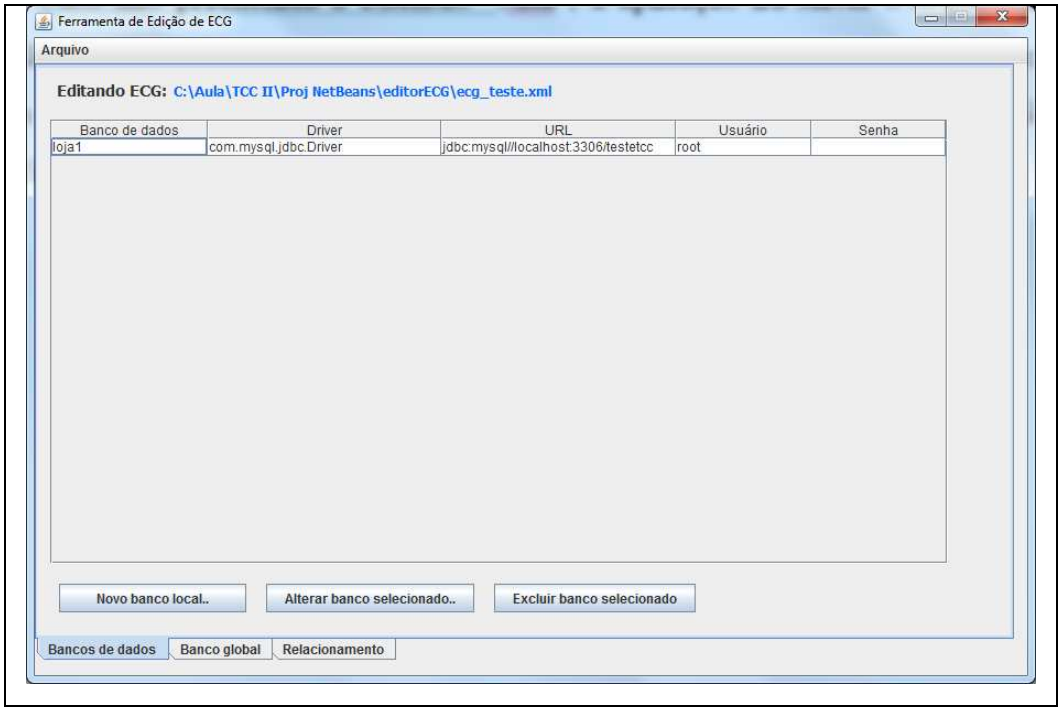


Figura 12 - Tela do cadastro de BDs após salvo ECG

Por fim, na aba “Relacionamento”, o *layout* da tela também sofreu algumas alterações. As figuras 13 e 14 fazem a comparação entre as versões.

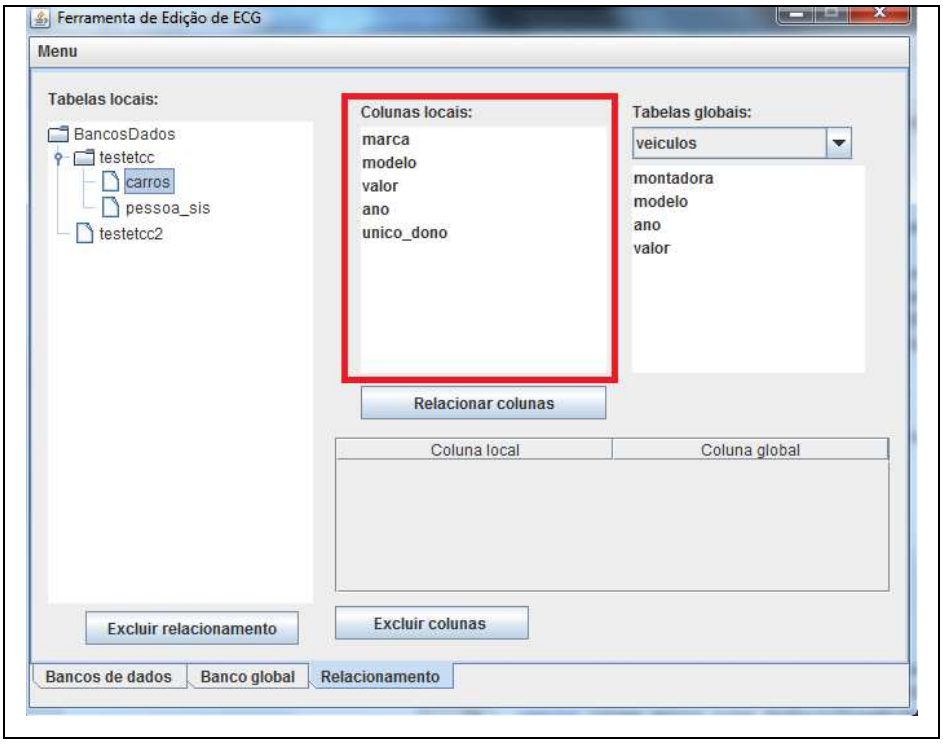


Figura 13 - Aba “Relacionamento” anterior

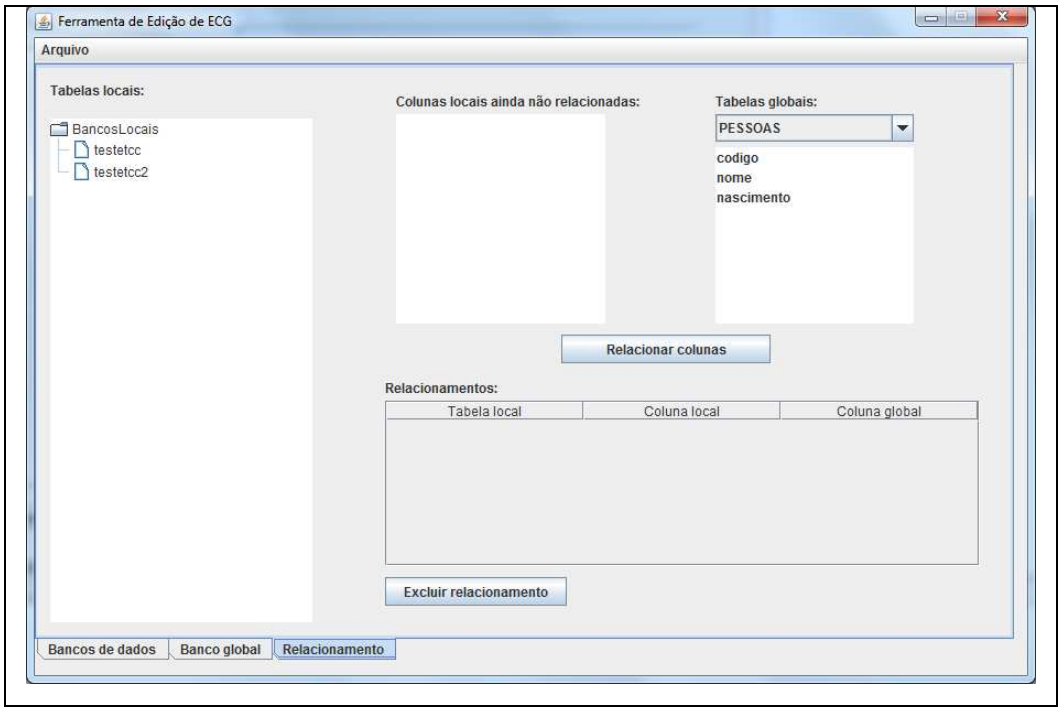


Figura 14 – Nova Aba “Relacionamento”

O botão “Excluir colunas” foi retirado, por não estar sendo utilizado. Foi inserida uma nova coluna na grade de relacionamentos, a coluna “Tabela local”. Também foi alterada a caixa de seleção de colunas locais: nesta agora só são listadas as colunas ainda não relacionadas, conforme a figura 15.

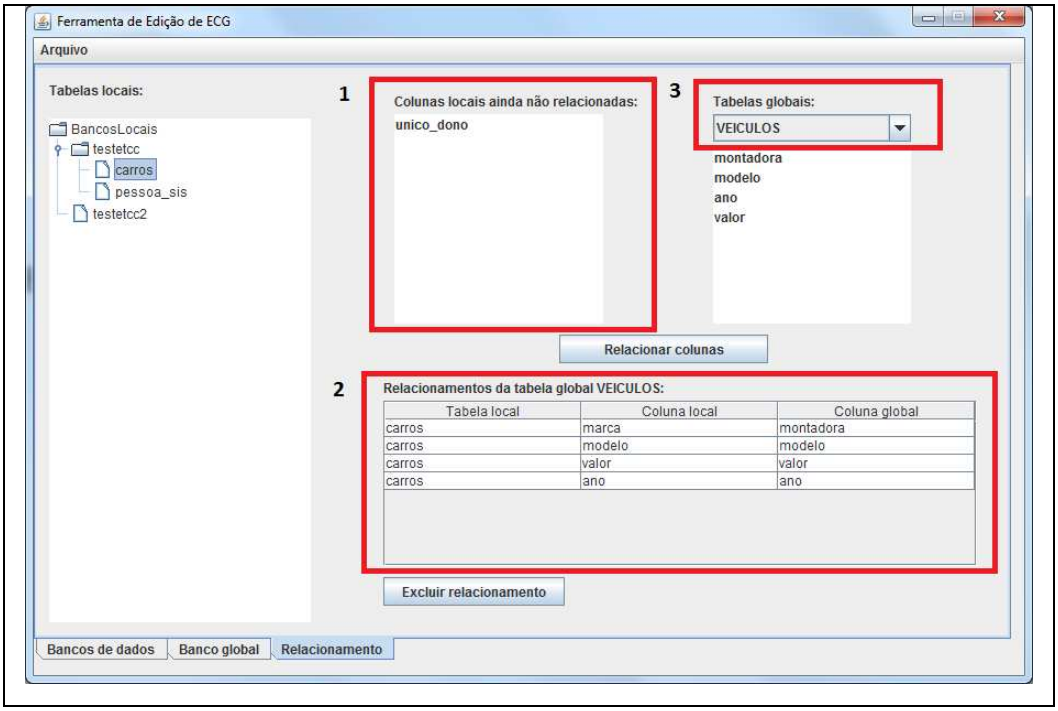


Figura 15 - Nova forma de inclusão de relacionamentos no ECG

Na figura 15 pode-se observar que somente a coluna `unico_dono` da tabela local `carros` ainda não foi relacionada com nenhuma coluna da tabela global `veiculos` (conforme

destaque 1 da figura 15). As demais colunas: marca, modelo, valor e ano já foram relacionadas com esta tabela global (conforme destaque 2 da figura 15). Na versão anterior, todas as colunas da tabela local permaneciam (conforme destaque na figura 13). Isto dificultava a visualização dos relacionamentos ainda não efetuados.

Agora também, conforme selecionada a tabela na árvore, a aplicação já seta a caixa de seleção na tabela global correspondente (conforme destaque 3 da figura 15) e lista os relacionamentos já existentes (conforme destaque 2 da figura 15).

3.3 ESTUDO DE CASO

Os testes referentes aos aperfeiçoamentos do *driver* foram feitos através da aplicação criada no trabalho de Gonçalves (2007): a aplicação “Informações Gerenciais”. Nesta aplicação é selecionado um arquivo XML de cadastro de ECG. Através de uma caixa de texto é possível digitar um comando SQL de consulta, sobre o esquema global selecionado. A aplicação então efetua a consulta através do *driver* e retorna as colunas e respectivos valores da consulta informada em uma grade de resultados.

Para poder realizar as consultas e efetuar testes em contexto global através do *driver*, foram criados dois sites, um utilizando banco MySql e outro com banco FireBird. Foram criadas tabelas em cada um desses sites com as mesmas características e finalidades, porém com nomes e colunas distintas, para exemplificar a diversidade que pode ocorrer entre os sites.

3.3.1 Modelo de Entidade Relacionamento (MER)

Para representar um ambiente real, pode-se tomar como exemplo um caso onde há duas videolocadoras de um mesmo dono, porém com sistemas distintos entre elas. O dono dessas locadoras deseja então fazer um *website* e que esse permita a consulta ao acervo de ambas videolocadoras.

A partir desse exemplo, foram criadas duas bases de dados simplificadas para exemplificar o caso citado, somente se referindo ao acervo de uma videolocadora. Vale frisar que não foram seguidos os padrões de normalização para criação das tabelas envolvidas e

também não foram incluídas todas as colunas necessárias em um ambiente real.

Na figura 16 é apresentado o MER do *site* criado através do banco MySql, denominado de `locadora_centro`.

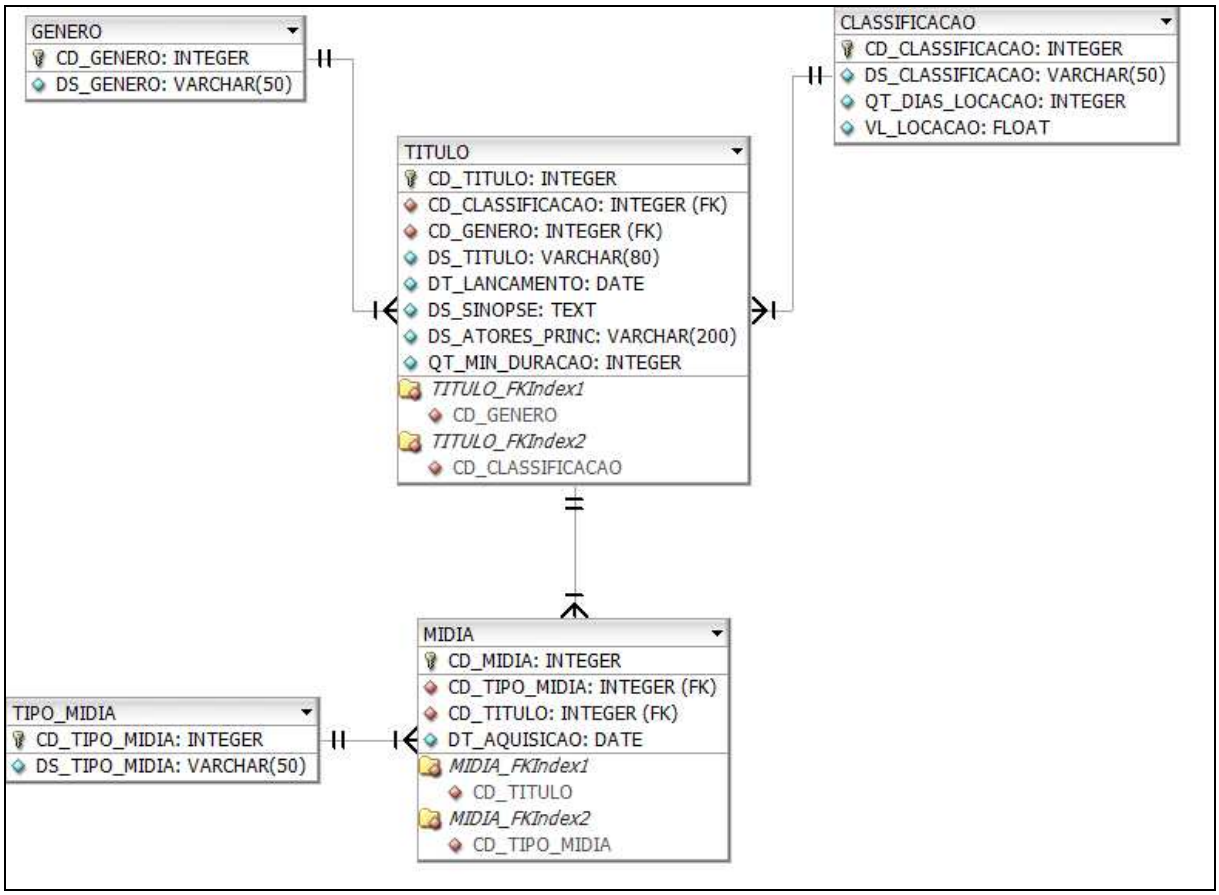


Figura 16 - MER do *site* MySql `locadora_centro`

Observa-se que nesta primeira base de dados foram criadas entidades para identificar o tipo de mídia (DVD, VHS, Blu-ray) e o gênero dos títulos de uma videolocadora (ação, suspense, romance, etc.).

Na figura 17 é apresentado o modelo de dados da segunda videolocadora, este *site* foi criado em um banco FireBird e denominado `locadora_interior`. Ao contrário do primeiro modelo apresentado, neste site não foram criadas entidades para tipo de mídia e gênero. Foram criadas apenas três entidades como pode ser observado na figura 17.

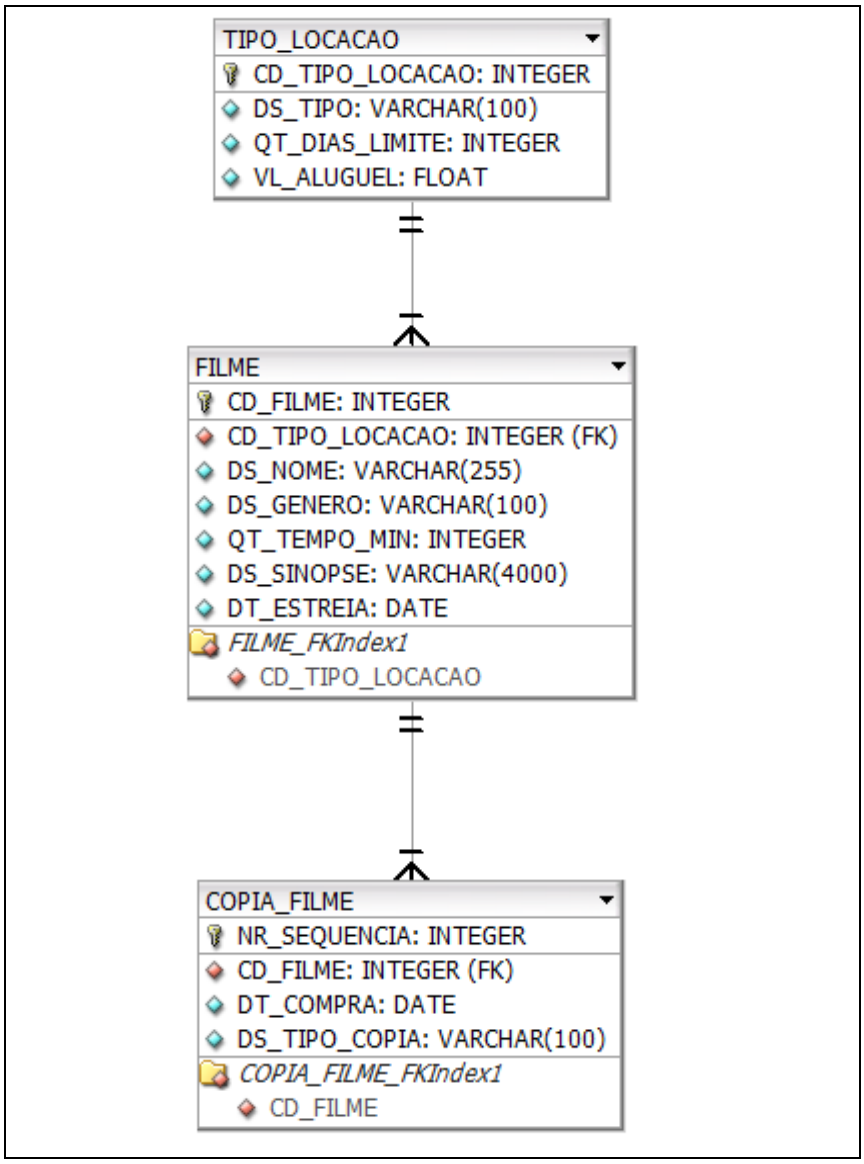


Figura 17 - MER do *site* FireBird `locadora_interior`

Porém, neste site optou-se por gravar o gênero do filme e o tipo de cópia, das entidades `FILME` e `COPIA_FILME` respectivamente, nas mesmas tabelas, através de domínios ou caixas de seleção. Com isso, no momento da criação do ECG, foi necessário adaptar estas colunas para que ficassem compatíveis com as tabelas `GENERO` e `CLASSIFICACAO` do primeiro *site*. Estas adaptações foram:

a) criar nova tabela global no ECG: `GL_GENERO`;

b) criar duas colunas nesta tabela global: `CD_GENERO` e `DS_GENERO`;

c) relacionar `CD_GENERO` da tabela local `GENERO` do site `locadora_centro` com `CD_GENERO` da tabela global `GL_GENERO`, o mesmo foi feito com a coluna `DS_GENERO`;

d) relacionar `CD_FILME` da tabela local `FILME` do site `locadora_interior` com `CD_GENERO` da tabela global `GL_GENERO` e a coluna local `DS_GENERO` de `FILME` com `DS_GENERO` de `GL_GENERO`.

Com esta adaptação nos relacionamentos do ECG puderam então ser feitas consultas em contexto global com junções entre as tabelas globais `GL_TITULO` e `GL_GENERO`.

3.3.2 ECG

O ECG criado para simular as consultas em contexto global e testar os recursos adicionados ao *driver* é representado no quadro 28. Nesta tabela são relacionadas todas as tabelas globais criadas no ECG e seus respectivos relacionamentos com as tabelas locais dos sites criados para o estudo de caso. O site MySql, que foi denominado `locadora_centro`, está representado na tabela seguinte na coluna “Site” com o número um. Já o site FireBird denominado `locadora_interior` está representado nesta mesma coluna com o número dois.

O XML gerado pela ferramenta de edição de ECG poderá ser consultado no Apêndice A.

Tabela global	Coluna global	Coluna local	Tabela local	Site
GL_GENERO	CD_GENERO	CD_GENERO	GENERO	1
		CD_FILME	FILME	2
	DS_GENERO	DS_GENERO	GENERO	1
		DS_GENERO	FILME	2
GL_CLASSIF	CD_CLASSIF	CD_CLASSIFICACAO	CLASSIFICACAO	1
		CD_TIPO_LOCACAO	TIPO_LOCACAO	2
	DS_CLASSIF	DS_CLASSIFICACAO	CLASSIFICACAO	1
		DS_TIPO_LOCACAO	TIPO_LOCACAO	2
	QT_DIAS_LOCACAO	QT_DIAS_LOCACAO	CLASSIFICACAO	1
		QT_DIAS_LIMITE	TIPO_LOCACAO	2
	VL_LOCACAO	VL_LOCACAO	CLASSIFICACAO	1
		VL_ALUGUEL	TIPO_LOCACAO	2
GL_TITULO	CD_TITULO	CD_TITULO	TITULO	1
		CD_FILME	FILME	2
	DS_TITULO	DS_TITULO	TITULO	1
		DS_NOME	FILME	2
	CD_GENERO	CD_GENERO	TITULO	1
		CD_FILME	FILME	2
	CD_CLASSIF	CD_CLASSIFICACAO	TITULO	1
		CD_TIPO_LOCACAO	FILME	2
	DT_LANCAMENTO	DT_LANCAMENTO	TITULO	1
		DT_ESTREIA	FILME	2
	DS_SINOPSE	DS_SINOPSE	TITULO	1
		DS_SINOPSE	FILME	2
	DS_ATORES_PRINC	DS_ATORES_PRINC	TITULO	1
	QT_MIN_DURACAO	QT_MIN_DURACAO	TITULO	1
		QT_TEMPO_MIN	FILME	2
GL_TIPO_VOLUME	CD_TIPO	CD_TIPO_MIDIA	TIPO_MIDIA	1
		NR_SEQUENICA	COPIA_FILME	2
	DS_TIPO	DS_TIPO_MIDIA	TIPO_MIDIA	1
		DS_TIPO_COPIA	COPIA_FILME	2
GL_VOLUME	CD_VOLUME	CD_MIDIA	MIDIA	1
		NR_SEQUENCIA	COPIA_FILME	2
	CD_TITULO	CD_TITULO	MIDIA	1
		CD_FILME	COPIA_FILME	2
	DT_AQUISICAO	DT_AQUISICAO	MIDIA	1
		DT_COMPRA	COPIA_FILME	2
	CD_TIPO	CD_TIPO_MIDIA	MIDIA	1
		NR_SEQUENCIA	COPIA_FILME	2

Quadro 28 - Relacionamentos do ECG criado para estudo de caso

3.3.3 Operacionalidade da implementação

Para poder realizar testes e validar as alterações foram povoadas as tabelas de ambos os sites. Nos quadros 29 e 30 são apresentados alguns comandos INSERT realizados em cada um dos *sites*.

```
-- Colunas (cd_genero,ds_genero)
insert into genero values (1,'Suspense');
insert into genero values (2,'Acao');
insert into genero values (3,'Aventura');
insert into genero values (4,'Terror');
insert into genero values (5,'Guerra');
insert into genero values (6,'Musical');
insert into genero values (7,'Infantil');

-- Colunas (cd_tipo_midia,ds_tipo_midia)
insert into tipo_midia values (1,'Dvd');
insert into tipo_midia values (2,'Blu-ray');

-- Colunas(cd_classificacao,ds_classificacao,qt_dias_locacao,vl_locacao)
insert into classificacao values (1,'Super lancamento',1,5.00);
insert into classificacao)values (4,'Lancamento',2,4.00);
insert into classificacao)values (10,'Vale a pena ver de novo',3,2.50);

-- Colunas (cd_titulo,cd_classificacao,cd_genero,ds_titulo,dt_lancamento,
-- ds_sinopse,ds_atores_princ,qt_min_duracao)
insert into titulo values (2,10,2,'Rambo 2','1990-05-03',
      'Mais uma aventura de John Rambo','Silvester Stallone',90);

insert into titulo values (8,4,2,'Duro de matar 4','2009-03-18',
      'Ninguem da conta do recado..','Bruce Willis',85);

insert into titulo values
      (35,4,1,'Pressagio','2009-04-10',
     'Em 1959, como parte das celebracoes de uma escola infantil, um grupo
     de alunos faz desenhos de como eles imaginam o futuro. Os desenhos
     ficarao guardados em uma capsula do tempo e serao abertos em 50
     anos.','Nicholas Cage, Rose Byrne',122);

insert into titulo values (42,1,1,'Sherlock Holmes','2010-02-05',
     'Em uma nova e dinâmica representação dos personagens mais famosos de
     Arthur Conan Doyle, Sherlock Holmes, enviará Holmes e o leal parceiro
     Watson em sua mais nova aventura.','Robert Downey Jr, Jude Law',130);

insert into values (45,1,3,'Avatar','2010-04-20',
     'Uma aventura na ilha de Pandora','Sigourney Weaver',150);

-- Colunas (cd_midia,cd_tipo_midia,cd_titulo,dt_aquisicao)
insert into midia values (1,1,2,'2008-08-15');
insert into midia values (5,1,8,'2009-03-20');
insert into midia values (10,1,35,'2009-04-15');
insert into midia values (15,1,35,'2009-04-20');
insert into midia values (32,1,42,'2009-02-06');
insert into midia values (33,2,42,'2009-02-06');
insert into midia values (40,1,45,'2008-08-15');
insert into midia values (41,2,45,'2008-08-15');
```

Quadro 29 - Povoamento das tabelas de locadora_centro

Os dados inseridos nas tabelas conforme os quadros 29 e 30 não foram importados mas inseridos manualmente, por este motivo são aleatórios e algumas das informações dos registros inseridos podem não ser verdadeiras. Após a inserção desses registros de forma manual foi criada uma rotina para povoar as bases de forma aleatória, para que houvesse uma quantidade de registros considerável para os testes.

```
-- Colunas (cd_tipo_locacao,ds_tipo,qt_dias_limite,vl_aluguel)
insert into tipo_locacao values (11,'Super lancamento',1,4.50);
insert into tipo_locacao values (12,'Lancamento',2,3.5);
insert into tipo_locacao values (13,'Antigos',3,2);

-- Colunas (cd_filme,cd_tipo_locacao,ds_nome,ds_genero,qt_tempo_min,
-- ds_sinopse,dt_estreia)
insert into filme values (15,13,'Loucademia de policia 4','Comedia',78,
    'Voce ira se divertir muito com os recrutas de Cel. Lacar..','1988-12-
18');

insert into filme values (16,13,'Top gun - Ases indonmáveis','Comedia',91,
    'Charlie Sheen voando baixo nessa comédia..','1993-10-15');

insert into filme values (20,12,'Pressagio','Suspense',91,
    'Em 1959, como parte das celebracoes de uma escola infantil, um grupo
    de alunos faz desenhos de como eles imaginam o futuro. Os desenhos
    ficarao guardados em uma capsula do tempo e serao abertos em 50
    anos.','2009-04-10');

insert into filme values(25,12,'Homem de Ferro','Acão',91,
    'O início do super-heroi da marvel','2009-02-20');

insert into filme values (31,11,'Avatar','Aventura',150,
    'Uma aventura na ilha de Pandora','2010-04-20');

-- Colunas (nr_sequencia,cd_filme,dt_compra,ds_tipo_copia)
insert into copia_filme values (1,15,'1988-01-05','Fita cassete');
insert into copia_filme values (65,15,'2006-08-12','Dvd');
insert into copia_filme values (80,16,'2005-06-14','Dvd');
insert into copia_filme values (81,20,'2009-05-8','Dvd');
insert into copia_filme values (85,25,'2009-03-02','Dvd');
insert into copia_filme values (90,31,'2010-05-02','Dvd');
insert into copia_filme values (91,31,'2010-05-02','Blu-ray');
```

Quadro 30 - Povoamento das tabelas de `locadora_interior`

Se analisados os quadros 29 e 30 se pode observar que há dois títulos repetidos entre os sites, “Presságio” e “Avatar”. Também houve nos sites, a inserção de mais de um volume do mesmo título. Ou seja, se for feita uma consulta através de ambos *sites* e não for utilizado agrupamento nesta, serão listados títulos repetidos. Dessa forma, estas inserções fornecem suporte para testes adequados de agrupamento global, em adição aos testes que podem ser feitos através de gênero e tipo de mídia, por exemplo.

A partir das tabelas locais povoadas e do ECG apresentado foi iniciado o processo de validação. Foi utilizada a mesma aplicação para validação utilizada por Gonçalves (2007), conforme citada anteriormente. Esta apenas acessa o *driver* e retorna em tela as consultas. Por

este motivo não foram necessárias alterações na mesma.

A demonstração da operacionalidade do *driver* será feita a seguir através de telas que exibem comandos `select` e seus resultados.

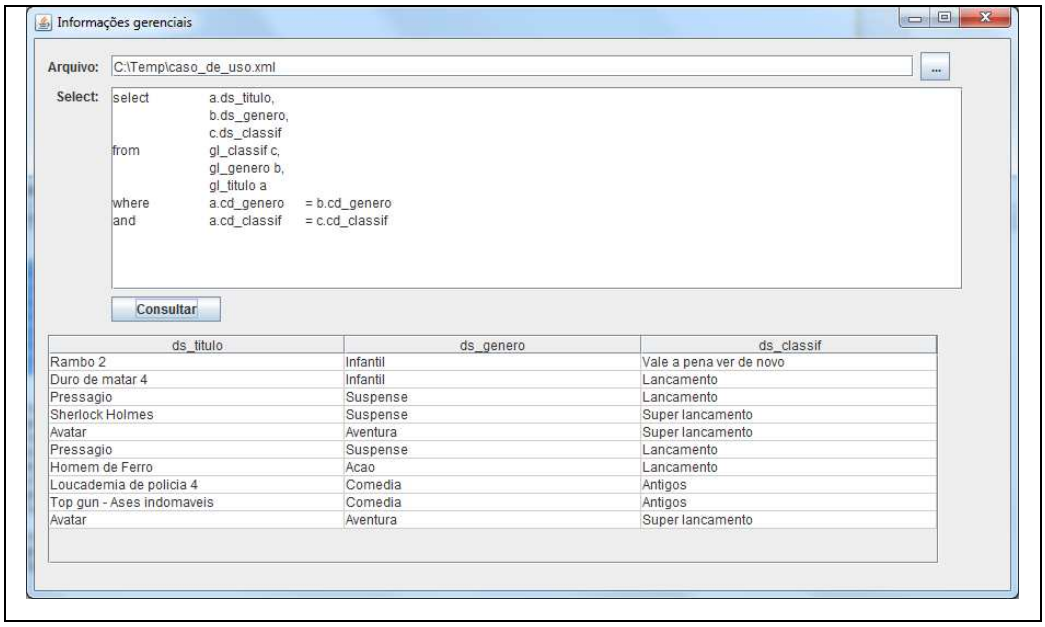


Figura 18 - Tela do sistema de validação (`SELECT` simples)

Na figura 18 foi feita uma consulta simples, sem a utilização de cláusulas de agrupamento ou ordenação e sem funções de agregação. Consultas nesse formato já eram possíveis em Gonçalves (2007). Este comando `select` da figura 16 retorna todos os títulos existentes em ambos os *sites*, e pode-se observar que alguns títulos (`ds_titulo`) se repetem.

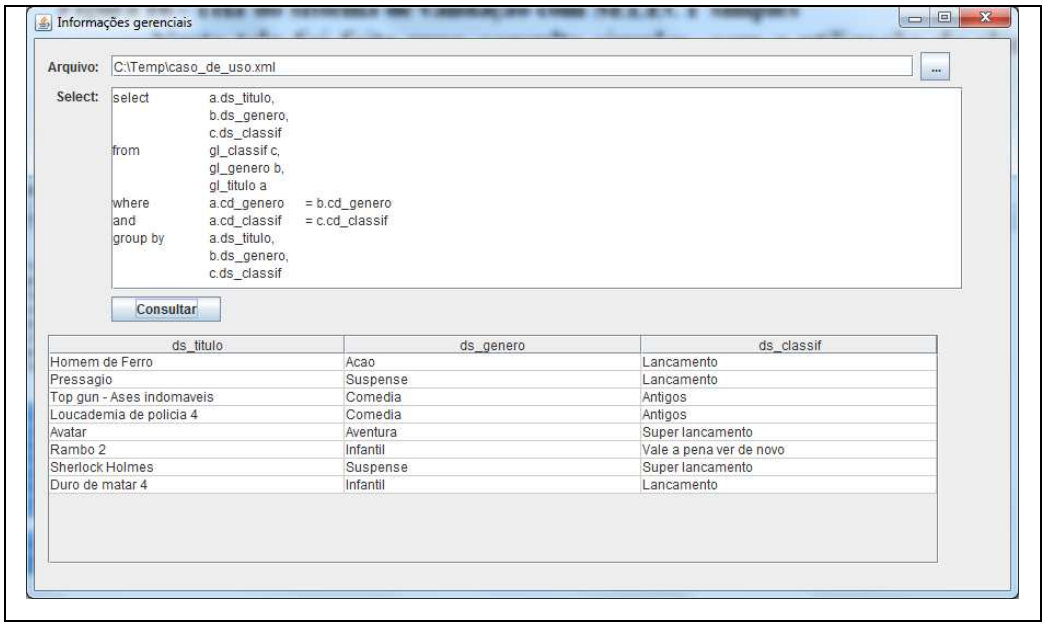


Figura 19 - Tela do sistema de validação (`SELECT` com agrupamento)

Já nesta tela, foi feita uma consulta com agrupamento, que também era possível em Gonçalves (2007). Porém, o agrupamento só era feito em cada um dos sites. Neste caso o

comando `select` feito na figura 19 traria o mesmo resultado do comando `select` da figura 16.

Na figura 20 foi criado um comando `select` para demonstrar algumas das funções de agregação em funcionamento.

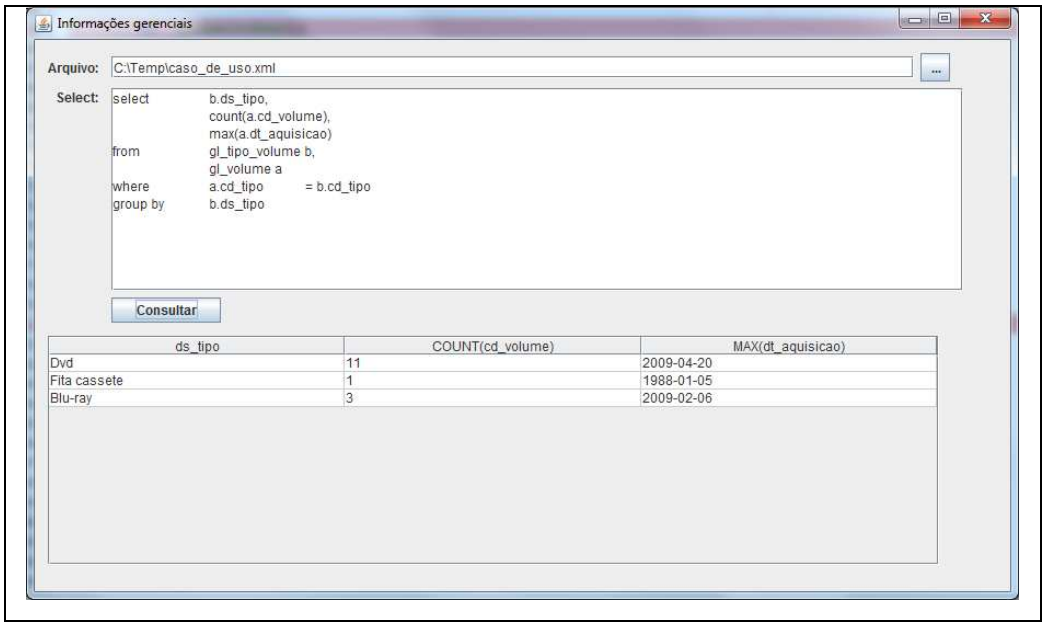


Figura 20 - Tela do sistema de Validação (`SELECT` com agregações)

A tela da figura 20 apresenta uma consulta para identificar quais os tipos de volumes, quantos há de cada tipo e a última aquisição de cada tipo. A figura 21 apresenta uma consulta utilizando restrição pós-agregação `HAVING`. Esta tela exibe uma consulta feita para obter quais títulos, que caso todos os volumes em ambas videolocadoras forem locados, somem R$10,00 ou mais.

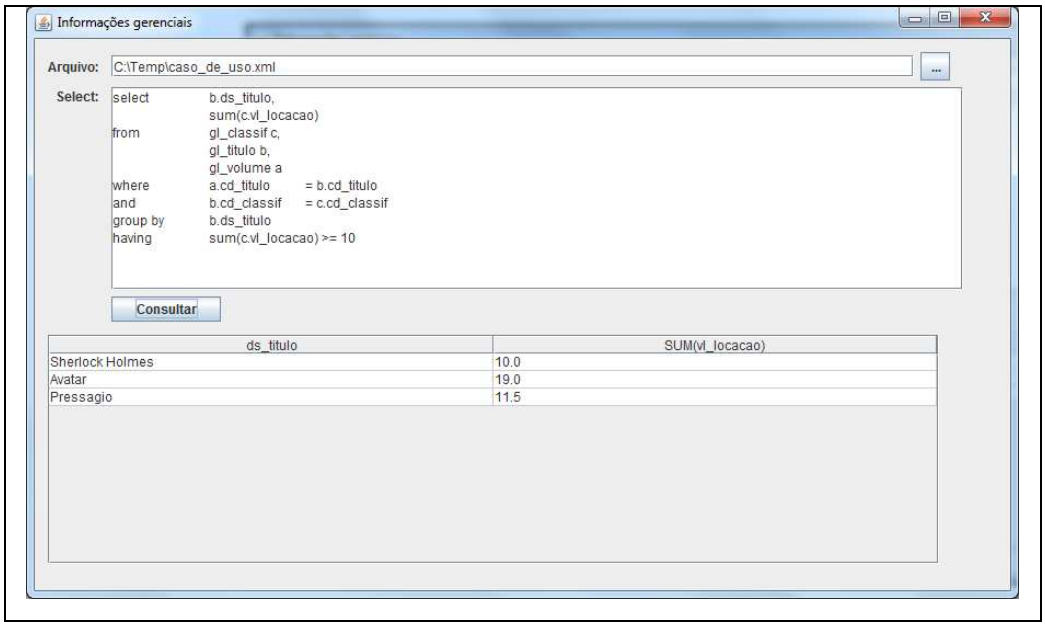


Figura 21 - Tela do sistema de validação (`SELECT` com restrição pós-agregação)

Por fim, na figura 22 tem-se a mesma consulta da figura 21, porém ordenada pelo nome do título.

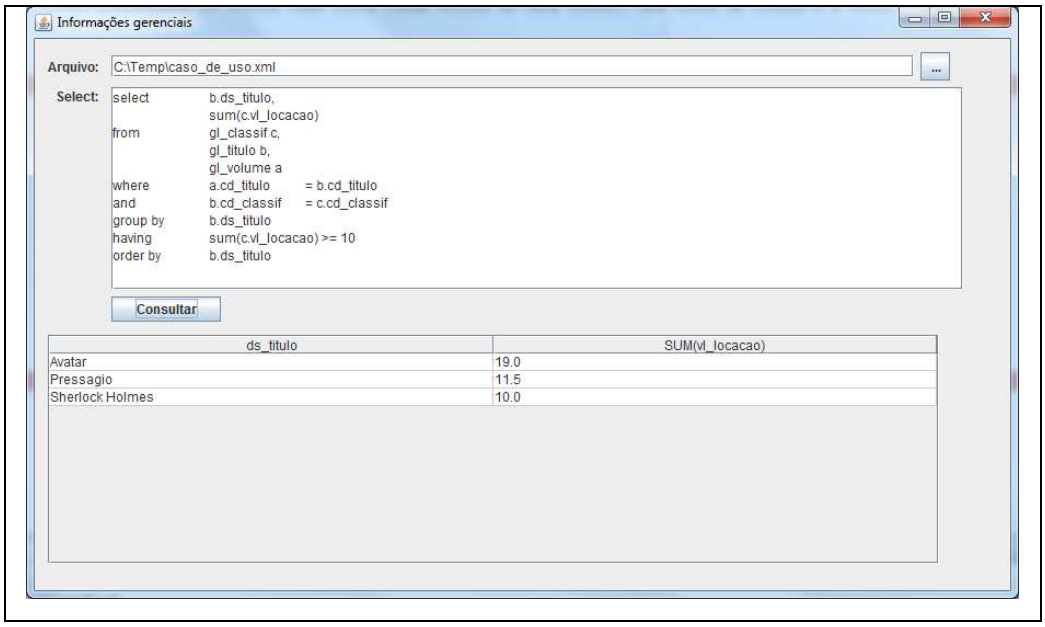


Figura 22 - Tela do sistema de validação (`SELECT` com ordenação)

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos com relação ao *driver* foram satisfatórios. Pode-se observar isto através das figuras 18, 19 e 20 do estudo de caso, que o agrupamento sobre contexto global funcionou corretamente. O funcionamento da ordenação em contexto global também pode ser visto através das figuras 21 e 22 do estudo de caso. Ainda há restrições para estes tipos de consulta, pois:

a) não é possível a utilização de funções de agregação que contenham fórmulas ou operações entre colunas, exemplo: `SUM(VALOR * 0.30)` e `SUM(VALOR + ACRESCIMOS);`
b) não é possível a ordenação através de apelido de coluna, exemplo: `SELECT MAX(VALOR) as MAIOR_VALOR FROM VALORES ORDER BY MAIOR_VALOR;`
c) não foram tratados casos onde a relação obtida na consulta tem tamanho superior a memória disponível;
d) não foi tratada a conversão de sintaxe entre SGDBs de diferentes fabricantes.

Para fazer uma análise de desempenho das rotinas de agrupamento e ordenação

implementadas foram realizadas as seguintes tarefas:

a) criado mais um *site* Oracle, com a mesma modelagem de dados do *site* MySql;
b) procurado na internet um acervo de filmes, totalizando 10446 filmes e criado um arquivo texto para cada gênero de filme à partir deste;
c) criação de uma aplicação para ler os arquivos textos e inserir em cada base de dados os filmes, separando por gênero entre os *sites*;
d) através da aplicação foram também incluídos mais de 18 mil registros na tabela `MIDIA` do site `locadora_centro` e 16 mil registros na tabela `COPIA_FILME` do site `locadora_interior`, por fim foram inseridos mais 35 mil registros no novo *site* Oracle;
e) registrado o tempo de determinadas consultas nas ferramentas de cada um dos SGBDs envolvidos;
f) incluído na aplicação Informações Gerenciais (utilizada para o estudo de caso) um *label* com o tempo de execução da consulta.

O laço de inserção de registros foi feito incrementando apenas a chave primária das tabelas locais `MIDIA` e `COPIA_FILME`, criando vários registros de mídia para cada filme, variando entre estes registros o tipo de mídia e a data de aquisição, ambos de forma randômica.

Nesta análise de desempenho foi possível identificar que há diferença de tempo entre a consulta local e global. Porém esta diferença é mais acentuada na consulta com ordenação. Isso porque a ordenação é feita somente sobre a relação global. No agrupamento e agregações, estes são feitos primeiro em cada um dos *sites* e só é processado então o resultado já agrupado de cada *site*, por este motivo tendo melhor desempenho.

Mesmo não havendo otimizações de consulta, foi feita uma avaliação de tempo de resposta de algumas consultas pertinentes ao trabalho, conforme quadros 31, 32, 33 e 34.

	Global	*Site* MySql	*Site* Oracle	*Site* FireBird
Comando SQL	`select count(*) from gl_volume`	`select count(*) from midia`	`select count(*) from midia`	`select count(*) from copia_filme`
Qtd. de registros	71404	18955	35606	16843
Tempo de resposta	0,083s	0,0573s	0,01s	0,022s

Quadro 31 - Análise de desempenho de agregação simples

	Global	*Site* MySql	*Site* Oracle	*Site* FireBird
Comando SQL	`select cd_titulo, max(dt_aquisicao) from gl_volume group by cd_titulo`	`select cd_titulo, max(dt_aquisicao) from midia group by cd_titulo`	`select cd_titulo, max(dt_aquisicao) from midia group by cd_titulo`	`select cd_filme, max(dt_compra) from copia_filme group by cd_filme`
Qtd. de registros	10446	2688	5323	2435
Tempo de resposta	0,439s	0,0396s	0,24s	0,11s

Quadro 32 - Análise de desempenho de agrupamento

	Global	*Site* MySql	*Site* Oracle	*Site* FireBird
Comando SQL	`select cd_titulo, ds_titulo from gl_titulo order by ds_titulo`	`select cd_titulo, ds_titulo from titulo order by ds_titulo`	`select cd_titulo, ds_titulo from titulo order by ds_titulo`	`select cd_filme, ds_nome from filme order by ds_nome`
Qtd. de registros	10446	2688	5323	2435
Tempo de resposta	0,311s	0,040s	0,20s	0,06s

Quadro 33 - Análise de desempenho de ordenação

	Global	*Site* MySql	*Site* Oracle	*Site* FireBird
Comando SQL	`select cd_titulo, count(*) from gl_volume group by cd_titulo having count(*) > 4`	`select cd_titulo, count(*) from midia group by cd_titulo having count(*) > 4`	`select cd_titulo, count(*) from midia group by cd_titulo having count(*) > 4`	`select cd_filme, count(*) from copia_filme group by cd_filme having count(*) > 4`
Qtd. de registros	7143	2054	3380	1709
Tempo de resposta	0,361s	0,01s	0,23s	0,0281s

Quadro 34 - Análise de desempenho de `HAVING`

Considerando os tempos de respostas das consultas acima apresentados, pode-se dizer que foram satisfatórios, pois a diferença de tempo entre a execução local e global não foi muito acentuada, mesmo não sendo otimizadas as consultas.

Quanto a Ferramenta de Edição de ECG, as alterações realizadas na interface contribuíram para dar maior usabilidade à ferramenta. A gravação e leitura dos arquivos XML de ECG ficaram mais simplificadas, tornou-se possível a edição de bancos previamente cadastrados no ECG e as descrições dos botões e títulos tornaram-se mais intuitivas.

Um ponto relevante quando a criação de ECGs é a possibilidade de tornar compatíveis tabelas com diferentes estruturas, como no exemplo das tabelas MIDIA e COPIA_FILME do estudo de caso. Apesar de apresentarem colunas com estruturas diferenciadas, onde uma contém a informação e outra contém uma referência, ambas puderam ser relacionadas com a mesma coluna da tabela global.

Em relação ao trabalho correlato de Gianisini (2006), um ponto em comum com este trabalho, é que ambos conseguem operar através de bancos de dados heterogêneos. A diferença porém é que Gianisini (2006) replica transações entre BDs locais e remotos, já Gonçalves (2007) permite consultas globais sobre BDs remotos. Gianisini (2006) necessitou utilizar o *framework Hibernate* para efetuar as transações entre os BDs. Este *framework* não foi necessário em Gonçalves (2007), por necessitar apenas da API JDBC para realização das consultas nos BDs.

Quanto ao trabalho de Bortolini (2004), apesar de não haver sido desenvolvida a ferramenta para replicação, foi feito um estudo aprofundado de fragmentação e foi utilizado para replicação o mesmo tipo de fragmentação utilizada no *driver* de Gonçalves (2007), a fragmentação horizontal.

O trabalho de Fink (2000) apresenta muitas semelhanças com o *driver* de Gonçalves (2007), porém não foi criado um ECG para relacionamento local e global e os comandos SQL não foram interpretados e convertidos.

4 CONCLUSÕES

O *driver* que foi continuado neste trabalho é baseado na ideia de unir informações de origens distintas. Isso é de grande valia para situações que se pode vivenciar hoje, como por exemplo, várias empresas de pequeno porte que são compradas por uma empresa de grande porte. Cada uma destas pequenas empresas possui um sistema próprio, podendo esses terem sido desenvolvidos com bancos de dados distintos entre eles. Porém, a empresa que adquiriu as demais, deseja ter um controle de forma total sobre algumas informações de seu grupo, como gastos, lucro e projetos em andamento, por exemplo. Ao invés de solicitar regularmente um relatório de cada uma dessas com estas informações, poderia ser desenvolvido um sistema utilizando o *driver* em questão e montando um ECG que reúne as informações desejadas de todas as empresas e de forma transparente.

As alterações realizadas no *driver* de Gonçalves (2007) foram fundamentais, para viabilizar a utilização de BD distribuído no contexto descrito anteriormente; pois apesar do mesmo já possibilitar a execução destas consultas, ele as fazia com algumas restrições no âmbito distribuído, como é o caso do agrupamento e ordenação. Porém ainda existem muitas coisas a serem exploradas neste *driver* para que a utilização do mesmo contemple todas as necessidades de um BDD.

Os objetivos de realizar consultas distribuídas com agrupamento e ordenação foram alcançados. Porém, há a restrição de que o resultado obtido na consulta pode ter tamanho superior à memória disponível no servidor do *driver*.

As alterações feitas na Ferramenta de Edição de ECG também foram de grande valia e tornaram mais fácil a construção e manutenção do ECG. Isto porque com as alterações realizadas, tornou-se possível alterar as configurações para os bancos locais já cadastrados, gravar e abrir XMLs de ECG mais facilmente e visualizar somente as colunas ainda pendentes de relacionamento.

Um dos objetivos deste trabalho, que era permitir a utilização da interface `PreparedStatement` não foi alcançado, ficando então para futuros trabalhos sobre o *driver*. A implementação deste objetivo foi iniciada, porém não concluída, devido ao *token* interrogação (que é utilizado pelo `PreparedStatement`) não constar na BNF de Gonçalves (2007). Sendo assim, teria que ser rescrita a BNF para dar continuidade a esta implementação e não houve tempo hábil para isto.

As rotinas desenvolvidas para agrupamento e ordenação das consultas distribuídas,

apesar de não serem extensas, necessitaram de muita pesquisa e testes para que fossem desenvolvidas e validadas de forma coerente. Outra dificuldade foi estudar e compreender toda a codificação já existente em Gonçalves (2007) para poder realizar os aperfeiçoamentos.

Existem também alguns itens que já haviam sido levantados por Gonçalves (2007), que também são sugestões de melhoria do *driver* e estão sendo listados novamente nas extensões.

Portanto, quanto mais for dada continuidade neste trabalho, maior será a possibilidade de utilização do *driver* em futuras aplicações, sejam acadêmicas, comerciais ou ainda com caráter social.

4.1 CONSIDERAÇÕES FINAIS

Após o desenvolvimento dos requisitos verificou-se que faltam ser implementados métodos para que o *driver* atenda a todas as funcionalidades existentes no *driver* JDBC padrão. Ou seja, fazer com que todos os métodos das interfaces JDBC sejam implementados corretamente. Por exemplo, a classe do *driver* que implementa a interface JDBC `Statement` não possui atualmente definidos os métodos `close`, `cancel`, `getResultSet`, entre outros.

Também apesar dos algoritmos pesquisados fazerem referência ao gerenciamento de memória e recursos, esses tratamentos não foram efetuados nas soluções apresentadas. As soluções propostas visam apenas o correto funcionamento das consultas, não havendo preocupação neste momento com otimização, situações extremas e prevenção de erros.

4.2 EXTENSÕES

Como sugestão para trabalhos futuros tem-se as seguintes:

a) possibilitar a utilização de comandos *Data Manipulation Language* (DML), como `INSERT`, `UPDATE` e `DELETE`;
b) desenvolvimento de técnicas para tratamentos de erros de conexão e falhas na comunicação entre os *sites*;
c) implementar a utilização de `JOIN` nas consultas globais;
d) implementar um recurso para que comandos `SELECT` específicos de um BD possam

ser mapeados e executados em outros BDs, atendendo suas particularidades;

e) possibilitar o uso de apelidos para definir as colunas da ordenação;
f) possibilitar o uso de fórmulas dentro das funções de agregação e funções de agregação composta;
g) possibilitar a eliminação de duplicatas (`DISTINCT`) em contexto global;
h) tratar situações extremas, onde o volume de dados obtidos seja superior a memória do servidor do BDD;
i) implementar heurísticas para otimização das consultas;
j) criar uma forma de definir chaves primárias para as tabelas globais;
k) permitir a criação de *Foreign Keys* (FK) globais e índices destas, para obter maior performance;
l) permitir suporte a outros idiomas na Ferramenta de Edição de ECG;
m) gerar MER do ECG, para facilitar visualização do mesmo.

Para grande parte dos itens acima citados pode ser utilizado como referência Garcia-Molina et al. (2001).

REFERÊNCIAS BIBLIOGRÁFICAS

BORTOLINI, César A. **Um protótipo de banco de dados distribuídos (caso Expresso São Miguel)**. 2004. 105 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Centro Tecnológico, Universidade Comunitária Regional de Chapecó, Chapecó. Disponível em: <http://www.unochapeco.edu.br/saa/tese/1326/tccII_CesarBortolini.pdf>. Acesso em: 08 jul. 2010.

DATE, Christopher J. **Introdução a sistemas de banco de dados**. Tradução Vandenberg D. de Souza, Publicare Consultoria e Serviços. 7. ed. Rio de Janeiro: Campus, 2000.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de bancos de dados**. 4. ed. São Paulo: Pearson Addison Wesley, 2005.

FINK, Ademir J. **Protótipo de Sistema para Acesso a Banco de Dados Distribuídos e Heterogêneos em Redes TCP/IP**. 2000. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GALS, **Documentação GALS**. [S.l.], [2010]. Disponível em: <http://gals.sourceforge.net/help.html>. Acesso em: 20 jun. 2010.

GARCIA-MOLINA, Ector; ULLMANN, Jefrey D.; WIDON, Jennifer. **Implementação de sistemas de bancos de dados**. Rio de Janeiro: Campus, 2001.

GIANISINI JÚNIOR, João B. **Desenvolvimento de um framework para replicação de dados entre bancos heterogêneos**. 2006. 101 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais. Universidade Regional de Blumenau, Blumenau.

GONÇALVES, Jacson. **Driver JDBC para consultas em bancos de dados distribuídos fragmentados horizontalmente**. 2007. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ORACLE. **Oracle oferece ampla solução de bancos de dados embutidos**. [S.l.], [2007]. Disponível em: <http://www.oracle.com/global/br/corporate/press/2007_jan/bancos_de_dados_embutidos.html>. Acesso em: 28 ago. 2009.

ÖZSU, Tamer; VALDURIEZ, Patrick. **Princípios de bancos de dados distribuídos**. Tradução Vandenberg D. de Souza. 2. ed. Rio de Janeiro: Campus, 2001.

RAMON, Fábio. **JDBC 2**: acesso a bancos de dados usando a linguagem Java. Guia de referência rápida. Rio de Janeiro: Novatec, 2001.

REESE, Geroge. **Database programming with JDBC and Java**. 2nd ed. Sebastopol: O' Reilly & Associates, 2000.

SILBERSCHATZ, Abraham; KORTH, Henry F; SUDARSHAN, Salim. **Sistema de banco de dados.** São Paulo: Pearson Education do Brasil, 1999.

VELOSO, René R. **Java e XML**: processamento de consultas XML com Java. Guia de consulta rápida. São Paulo: Novatec, 2003.

WHITE, Seth et al. **JDBC API tutorial and reference**: universal data access for the Java 2 platform. 2nd ed. California: Addison Wesley, 1999.

APÊNDICE A – Arquivo XML utilizado para estudo de caso

No quadro 35 é apresentado o arquivo XML gerado pela ferramenta de edição de ECG. Este é o ECG que foi criado para validações e o estudo de caso.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecg>
   <bancosDados>
      <bancoDado>
         <nome>locadora_centro</nome>
         <driver>com.mysql.jdbc.Driver</driver>
         <url>jdbc:mysql://localhost:3306/locadora_centro</url>
         <usuario>root</usuario>
         <senha/>
      </bancoDado>
      <bancoDado>
         <nome>locadora_interior</nome>
         <driver>org.firebirdsql.jdbc.FBDriver</driver>
         <url>
              jdbc:firebirdsql:localhost/3050:C:\\Aula\\TCC II\\LOCADORA_INTERIOR.GDB
         </url>
         <usuario>SYSDBA</usuario>
         <senha>masterkey</senha>
      </bancoDado>
   </bancosDados>
   <tabelasGlobais>
      <tabelaGlobal>
         <nome>GL_VOLUME</nome>
         <colunas>
            <coluna_0>CD_VOLUME</coluna_0>
            <coluna_1>CD_TITULO</coluna_1>
            <coluna_2>DT_AQUISICAO</coluna_2>
            <coluna_3>CD_TIPO</coluna_3>
         </colunas>
      </tabelaGlobal>
      <tabelaGlobal>
         <nome>GL_GENERO</nome>
         <colunas>
            <coluna_0>CD_GENERO</coluna_0>
            <coluna_1>DS_GENERO</coluna_1>
         </colunas>
      </tabelaGlobal>
      <tabelaGlobal>
         <nome>GL_CLASSIF</nome>
         <colunas>
            <coluna_0>CD_CLASSIF</coluna_0>
            <coluna_1>DS_CLASSIF</coluna_1>
            <coluna_2>QT_DIAS_LOCACAO</coluna_2>
            <coluna_3>VL_LOCACAO</coluna_3>
         </colunas>
      </tabelaGlobal>
      <tabelaGlobal>
         <nome>GL_TIPO_VOLUME</nome>
         <colunas>
            <coluna_0>CD_TIPO</coluna_0>
            <coluna_1>DS_TIPO</coluna_1>
         </colunas>
      </tabelaGlobal>
      <tabelaGlobal>
         <nome>GL_TITULO</nome>
         <colunas>
            <coluna_0>CD_TITULO</coluna_0>
            <coluna_1>DS_TITULO</coluna_1>
            <coluna_2>CD_GENERO</coluna_2>
            <coluna_3>CD_CLASSIF</coluna_3>
            <coluna_4>DT_LANCAMENTO</coluna_4>
```

```
            <coluna_5>DS_SINOPSE</coluna_5>
            <coluna_6>DS_ATORES_PRINC</coluna_6>
            <coluna_7>QT_MIN_DURACAO</coluna_7>
         </colunas>
      </tabelaGlobal>
   </tabelasGlobais>
   <relacionamentos>
      <relacionamento>
         <nomeBanco>locadora_centro</nomeBanco>
         <tabelas>
            <tabela>
               <nomeTabela>
                  <nomeGlobal>GL_GENERO</nomeGlobal>
                  <nomeLocal>genero</nomeLocal>
               </nomeTabela>
               <colunas>
                  <coluna>
                     <nomeColunaGlobal>CD_GENERO</nomeColunaGlobal>
                     <nomeColunaLocal>CD_GENERO</nomeColunaLocal>
                  </coluna>
                  <coluna>
                     <nomeColunaGlobal>DS_GENERO</nomeColunaGlobal>
                     <nomeColunaLocal>DS_GENERO</nomeColunaLocal>
                  </coluna>
               </colunas>
            </tabela>
            <tabela>
               <nomeTabela>
                  <nomeGlobal>GL_TIPO_VOLUME</nomeGlobal>
                  <nomeLocal>tipo_midia</nomeLocal>
               </nomeTabela>
               <colunas>
                  <coluna>
                     <nomeColunaGlobal>CD_TIPO</nomeColunaGlobal>
                     <nomeColunaLocal>CD_TIPO_MIDIA</nomeColunaLocal>
                  </coluna>
                  <coluna>
                     <nomeColunaGlobal>DS_TIPO</nomeColunaGlobal>
                     <nomeColunaLocal>DS_TIPO_MIDIA</nomeColunaLocal>
                  </coluna>
               </colunas>
            </tabela>
            <tabela>
               <nomeTabela>
                  <nomeGlobal>GL_CLASSIF</nomeGlobal>
                  <nomeLocal>classificacao</nomeLocal>
               </nomeTabela>
               <colunas>
                  <coluna>
                     <nomeColunaGlobal>CD_CLASSIF</nomeColunaGlobal>
                     <nomeColunaLocal>CD_CLASSIFICACAO</nomeColunaLocal>
                  </coluna>
                  <coluna>
                     <nomeColunaGlobal>DS_CLASSIF</nomeColunaGlobal>
                     <nomeColunaLocal>DS_CLASSIFICACAO</nomeColunaLocal>
                  </coluna>
                  <coluna>
                     <nomeColunaGlobal>QT_DIAS_LOCACAO</nomeColunaGlobal>
                     <nomeColunaLocal>QT_DIAS_LOCACAO</nomeColunaLocal>
                  </coluna>
                  <coluna>
                     <nomeColunaGlobal>VL_LOCACAO</nomeColunaGlobal>
                     <nomeColunaLocal>VL_LOCACAO</nomeColunaLocal>
                  </coluna>
               </colunas>
            </tabela>
            <tabela>
               <nomeTabela>
                  <nomeGlobal>GL_TITULO</nomeGlobal>
                  <nomeLocal>titulo</nomeLocal>
               </nomeTabela>
```

```
            <colunas>
               <coluna>
                  <nomeColunaGlobal>CD_TITULO</nomeColunaGlobal>
                  <nomeColunaLocal>CD_TITULO</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>DS_TITULO</nomeColunaGlobal>
                  <nomeColunaLocal>DS_TITULO</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>CD_GENERO</nomeColunaGlobal>
                  <nomeColunaLocal>CD_GENERO</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>CD_CLASSIF</nomeColunaGlobal>
                  <nomeColunaLocal>CD_CLASSIFICACAO</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>DT_LANCAMENTO</nomeColunaGlobal>
                  <nomeColunaLocal>DT_LANCAMENTO</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>DS_SINOPSE</nomeColunaGlobal>
                  <nomeColunaLocal>DS_SINOPSE</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>DS_ATORES_PRINC</nomeColunaGlobal>
                  <nomeColunaLocal>DS_ATORES_PRINC</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>QT_MIN_DURACAO</nomeColunaGlobal>
                  <nomeColunaLocal>QT_MIN_DURACAO</nomeColunaLocal>
               </coluna>
            </colunas>
         </tabela>
         <tabela>
            <nomeTabela>
               <nomeGlobal>GL_VOLUME</nomeGlobal>
               <nomeLocal>midia</nomeLocal>
            </nomeTabela>
            <colunas>
               <coluna>
                  <nomeColunaGlobal>CD_VOLUME</nomeColunaGlobal>
                  <nomeColunaLocal>CD_MIDIA</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>CD_TITULO</nomeColunaGlobal>
                  <nomeColunaLocal>CD_TITULO</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>CD_TIPO</nomeColunaGlobal>
                  <nomeColunaLocal>CD_TIPO_MIDIA</nomeColunaLocal>
               </coluna>
               <coluna>
                  <nomeColunaGlobal>DT_AQUISICAO</nomeColunaGlobal>
                  <nomeColunaLocal>DT_AQUISICAO</nomeColunaLocal>
               </coluna>
            </colunas>
         </tabela>
      </tabelas>
   </relacionamento>
   <relacionamento>
      <nomeBanco>locadora_interior</nomeBanco>
      <tabelas>
         <tabela>
            <nomeTabela>
               <nomeGlobal>GL_CLASSIF</nomeGlobal>
               <nomeLocal>TIPO_LOCACAO</nomeLocal>
            </nomeTabela>
            <colunas>
               <coluna>
```

```
                <nomeColunaGlobal>CD_CLASSIF</nomeColunaGlobal>
                <nomeColunaLocal>CD_TIPO_LOCACAO</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>DS_CLASSIF</nomeColunaGlobal>
                <nomeColunaLocal>DS_TIPO</nomeColunaLocal>
            </coluna>
            <coluna>
              <nomeColunaGlobal>QT_DIAS_LOCACAO</nomeColunaGlobal>
                <nomeColunaLocal>QT_DIAS_LIMITE</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>VL_LOCACAO</nomeColunaGlobal>
                <nomeColunaLocal>VL_ALUGUEL</nomeColunaLocal>
            </coluna>
        </colunas>
    </tabela>
    <tabela>
        <nomeTabela>
            <nomeGlobal>GL_TITULO</nomeGlobal>
            <nomeLocal>FILME</nomeLocal>
        </nomeTabela>
        <colunas>
            <coluna>
                <nomeColunaGlobal>CD_TITULO</nomeColunaGlobal>
                <nomeColunaLocal>CD_FILME</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>DS_TITULO</nomeColunaGlobal>
                <nomeColunaLocal>DS_NOME</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>CD_CLASSIF</nomeColunaGlobal>
                <nomeColunaLocal>CD_TIPO_LOCACAO</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>CD_GENERO</nomeColunaGlobal>
                <nomeColunaLocal>CD_FILME</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>DT_LANCAMENTO</nomeColunaGlobal>
                <nomeColunaLocal>DT_ESTREIA</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>DS_SINOPSE</nomeColunaGlobal>
                <nomeColunaLocal>DS_SINOPSE</nomeColunaLocal>
            </coluna>
        </colunas>
    </tabela>
    <tabela>
        <nomeTabela>
            <nomeGlobal>GL_GENERO</nomeGlobal>
            <nomeLocal>FILME</nomeLocal>
        </nomeTabela>
        <colunas>
            <coluna>
                <nomeColunaGlobal>DS_GENERO</nomeColunaGlobal>
                <nomeColunaLocal>DS_GENERO</nomeColunaLocal>
            </coluna>
            <coluna>
                <nomeColunaGlobal>CD_GENERO</nomeColunaGlobal>
                <nomeColunaLocal>CD_FILME</nomeColunaLocal>
            </coluna>
        </colunas>
    </tabela>
    <tabela>
        <nomeTabela>
            <nomeGlobal>GL_VOLUME</nomeGlobal>
            <nomeLocal>COPIA_FILME</nomeLocal>
        </nomeTabela>
        <colunas>
```

```
                <coluna>
                   <nomeColunaGlobal>CD_VOLUME</nomeColunaGlobal>
                   <nomeColunaLocal>NR_SEQUENCIA</nomeColunaLocal>
                </coluna>
                <coluna>
                   <nomeColunaGlobal>CD_TITULO</nomeColunaGlobal>
                   <nomeColunaLocal>CD_FILME</nomeColunaLocal>
                </coluna>
                <coluna>
                   <nomeColunaGlobal>DT_AQUISICAO</nomeColunaGlobal>
                   <nomeColunaLocal>DT_COMPRA</nomeColunaLocal>
                </coluna>
                <coluna>
                   <nomeColunaGlobal>CD_TIPO</nomeColunaGlobal>
                   <nomeColunaLocal>NR_SEQUENCIA</nomeColunaLocal>
                </coluna>
             </colunas>
          </tabela>
          <tabela>
             <nomeTabela>
                <nomeGlobal>GL_TIPO_VOLUME</nomeGlobal>
                <nomeLocal>COPIA_FILME</nomeLocal>
             </nomeTabela>
             <colunas>
                <coluna>
                   <nomeColunaGlobal>CD_TIPO</nomeColunaGlobal>
                   <nomeColunaLocal>NR_SEQUENCIA</nomeColunaLocal>
                </coluna>
                <coluna>
                   <nomeColunaGlobal>DS_TIPO</nomeColunaGlobal>
                   <nomeColunaLocal>DS_TIPO_COPIA</nomeColunaLocal>
                </coluna>
             </colunas>
          </tabela>
       </tabelas>
    </relacionamento>
 </relacionamentos>
</ecg>
```

Quadro 35 - Conteúdo do XML do ECG utilizado no estudo de caso

APÊNDICE B – BNF SQL utilizada no GALS

No quadro 36 é apresentada a BNF que foi utilizada para geração das classes analisadoras pelo GALS em Gonçalves (2007).

```
#Options
GenerateScanner = true
GenerateParser = true
Language = Java
ScannerName = Lexico
ParserName = Sintatico
SemanticName = Semantico
Package = gals
ScannerCaseSensitive = false
ScannerTable = Hardcode
Input = String
Parser = LL
#RegularDefinitions
Digit     : [0-9]
minuscula : [a-z]
maiuscula : [A-Z]
UnderLine : [_]
Letter    : {maiuscula}|{minuscula}|{UnderLine}
noQuote   : [^"'"\n\r]
COMMENT   : "/*"[^"*"]*"*/"
WS      : [\s\t\n\r]
#Tokens
// outros
ident      : {Letter} ( {Letter} | {Digit} )*
integer_   : {Digit} ( {Digit} )*
float_     : {Digit} {Digit}* ("." {Digit} {Digit}*)?
SQLString  : "'" {noQuote}* "'"

// Palavras reservadas
UNION          = ident : "UNION"
ALL            = ident : "ALL"
FROM           = ident : "FROM"
SELECT         = ident : "SELECT"
DISTINCT              = ident : "DISTINCT"
AS             = ident : "AS"
WHERE          = ident : "WHERE"
HAVING         = ident : "HAVING"
ORDER          = ident : "ORDER"
GROUP          = ident : "GROUP"
BY             = ident : "BY"
UPPER          = ident : "UPPER"
MONTH          = ident : "MONTH"
YEAR           = ident : "YEAR"
COUNT          = ident : "COUNT"
```

```
SUM             = ident : "SUM"
MAX             = ident : "MAX"
MIN             = ident : "MIN"
AVG             = ident : "AVG"
NULL            = ident : "NULL"
DESC            = ident : "DESC"
ASC             = ident : "ASC"
NOT             = ident : "NOT"
AND             = ident : "AND"
OR              = ident : "OR"
LIKE            = ident : "LIKE"
IS              = ident : "IS"
BETWEEN         = ident : "BETWEEN"
IN              = ident : "IN"

// Símbolos especiais
"("
")"
"*"
"+"
","
"-"
"."
"/"
":"
";"
"<"
"="
">"
"<>"
"<="
">="
"'"

// Ignorar
: {WS}*
:! {COMMENT}
#NonTerminals
<SQLTest>
<Comando>
<PontoVirgula>
<SelectSQL>
<UnionSelect>
<AllOpcional>
<SubSelectSQL>
<WhereOpcional>
<SelectStmt>
<OrderByClauseOpcional>
<HavingClauseOpcional>
<GroupByClauseOpcional>
<SelectClause>
```

```
<DistinctAllOpcional>
<FromClause>
<FromTableList>
<QualifiedSeparator>
<QualifiedTable>
<AsAliasOpcional>
<WhereClause>
<HavingClause>
<OrderByClause>
<GroupByClause>
<SelectFieldList>
<SeparatorSelectField>
<SelectField>
<AsAlias>
<FunctionExpr>
<SeparatorExpression>
<W>
<ColumnFunction>
<AsteriscoDistinct>
<DistinctExpression>
<DistinctOpcional>
<Function>
<ColumnList>
<SeparatorColumnName>
<ColumnName>
<PontoCampo>
<IdentAsterisco>
<FieldList>
<SeparatorField>
<Field>
<Null>
<Alias>
<OrderByFldList>
<SeparatorOrderBy>
<OrderByField>
<ColumnInteger>
<OrdemOpcional>
<SearchCondition>
<Expression>
<RelationSimpleExp>
<SimpleExpression>
<RepeteOperator>
<NotOpcional>
<Term>
<K>
<OpenExpClose>
<ExpSubSel>
<FieldTest>
<TestOpcional>
<MenosOpcional>
<Param>
```

```
<NotOperator>
<Operator>
<MathOperator>
<WordOperator>
<LikeTest>
<SqlParam>
<NullTest>
<Relation>
<TestExpr>
<L>
<BetweenExpr>
<InSetExpr>
<FieldSub>
<IntOpcional>
<ItemSeparator>
<OpenParens>
<CloseParens>
#Grammar
<SQLTest>             ::=  <Comando>  <PontoVirgula>;
<PontoVirgula>        ::= ";" | î ;
<Comando>             ::= <SelectSQL>;
<SelectSQL>           ::=  <SelectStmt> <UnionSelect>;
<UnionSelect>         ::= UNION <AllOpcional> <SelectStmt> <UnionSelect> | î ;
<AllOpcional>         ::= ALL | î ;
<SelectStmt>          ::= <SelectClause> <FromClause> <WhereOpcional>
<GroupByClauseOpcional><HavingClauseOpcional> <OrderByClauseOpcional>;
<SelectClause> ::= SELECT #1 <DistinctAllOpcional> <SelectFieldList>;
<DistinctAllOcional>  ::= DISTINCT | ALL | î ;
<Null>                ::= NULL;
<SelectFieldList>     ::= <SelectField> <SeparatorSelectField>;
<SeparatorSelectField>::= <ItemSeparator> <SelectField> <SeparatorSelectField> | î ;
<SelectField>         ::= <Expression> #2 <AsAlias> | "*" #2;
<AsAlias>             ::= AS <Alias> #3|  î ;
<Alias>               ::= SQLString | ident;
<Expression>          ::= <SimpleExpression> <RelationSimpleExp>;
<RelationSimpleExp>   ::= <Relation> <SimpleExpression> <RelationSimpleExp> |  î ;
<SimpleExpression>    ::= <NotOpcional> <Term> <RepeteOperator>;
<RepeteOperator>      ::= <Operator> <NotOpcional> <Term> <RepeteOperator> |  î ;
<NotOpcional>         ::= <NotOperator> |  î;
<Term>                ::= <MenosOpcional> <K>;
<K>                   ::= <FieldTest> | <ColumnFunction> | <FunctionExpr> | <OpenExpClose>;
<OpenExpClose>        ::= <OpenParens> <ExpSubSel> <CloseParens>;
<ExpSubSel>           ::= <Expression> | <SubSelectSQL>;
<SubSelectSQL>        ::= <SelectSQL>;
<FieldTest>           ::= <Field> <TestOpcional>;
<TestOpcional>        ::= <TestExpr> |  î;
<NotOperator>         ::= NOT;
<MenosOpcional>              ::= "-" | î;
<ItemSeparator>              ::= ",";
<OpenParens>          ::= "(";
<CloseParens>         ::= ")";
```

```
<NullTest>            ::= IS <NotOpcional>  <Null>;
<Relation>            ::= "=" | "<>"| "<" | "<=" | ">" | ">=";
<TestExpr>            ::= <NullTest> | <NotOpcional> <L>;
<FieldList>           ::= <Field> <SeparatorField>;
<SeparatorField>      ::= <ItemSeparator> <Field> <SeparatorField> |  î;
<Field>               ::= <ColumnName> | <Null> | float_ | integer_ | SQLString |":" ident ;
<ColumnName>          ::= ident #5 <PontoCampo>;
<PontoCampo>          ::= "." <IdentAsterisco> |  î;
<IdentAsterisco>      ::= ident #2 | "*";
<L>                   ::= <InSetExpr> | <BetweenExpr> | <LikeTest>;
<LikeTest>            ::= LIKE <SqlParam>;
<BetweenExpr>         ::= BETWEEN <Field> AND <Field>;
<InSetExpr>           ::= IN <OpenParens> <FieldSub> <CloseParens>;
<FieldSub>            ::= <FieldList> | <SubSelectSQL>;
<SqlParam>            ::= SQLString | <Param>;
<Operator>            ::= <MathOperator> | <WordOperator>;
<MathOperator>        ::= "*" | "/"| "+"| "-";
<WordOperator>        ::= AND | OR;
<FunctionExpr>        ::= <W> <OpenParens> <Expression> <SeparatorExpression> <CloseParens>;
<SeparatorExpression> ::= <ItemSeparator> <Expression> <SeparatorExpression> |  î;
<W>                   ::= UPPER| MONTH| YEAR;
<ColumnFunction>      ::= <Function> <OpenParens> <AsteriscoDistinct> <CloseParens>;
<AsteriscoDistinct>   ::= "*"| <DistinctExpression>;
<DistinctExpression>  ::= <DistinctOpcional> <Expression> ;
<DistinctOpcional>    ::= DISTINCT | î;
<Function>            ::= COUNT | SUM | MAX | MIN |AVG;
<ColumnList>          ::= <ColumnName> <SeparatorColumnName>;
<SeparatorColumnName>  ::= <ItemSeparator> <ColumnName> <SeparatorColumnName> | î;
<FromClause>          ::= FROM #9 <FromTableList>;
<FromTableList>               ::= <QualifiedTable> <QualifiedSeparator>;
<QualifiedTable>      ::= ident #4 <AsAliasOpcional>;
<AsAliasOpcional>     ::= <Alias> #6 | î;
<QualifiedSeparator>  ::= "," <FromTableList> | î;
<WhereOpcional>               ::= <WhereClause> | î;
<WhereClause>         ::= WHERE #10 <SearchCondition>;
<SearchCondition>     ::= <Expression>;
<GroupByClauseOpcional> ::= <GroupByClause> | î;
<GroupByClause>               ::= GROUP BY #11 <FieldList>;
<HavingClauseOpcional>::= <HavingClause> | î;
<HavingClause>        ::= HAVING #12 <SearchCondition>;
<OrderByClauseOpcional> ::= <OrderByClause> | î;
<OrderByClause>               ::= ORDER BY #13 <OrderByFldList>;
<OrderByFldList>      ::= <OrderByField> <SeparatorOrderBy>;
<SeparatorOrderBy>    ::= <ItemSeparator> <OrderByField> <SeparatorOrderBy> |  î;
<OrderByField>        ::= <ColumnInteger> <OrdemOpcional>;
<ColumnInteger>               ::= <ColumnName> | integer_ ;<OrdemOpcional>       ::= DESC |
ASC |  î;
```

Quadro 36 - BNF da linguagem SQL utilizada no GALS