Francisco Roeder

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

Orientador:
Prof. Adilson Vahldick

Roteiro da apresentação

- Introdução
- Objetivos
- Fundamentação teórica
- Trabalhos correlatos
- Especificação
- Desenvolvimento
- Considerações finais
- Conclusão
- Extensões

Introdução

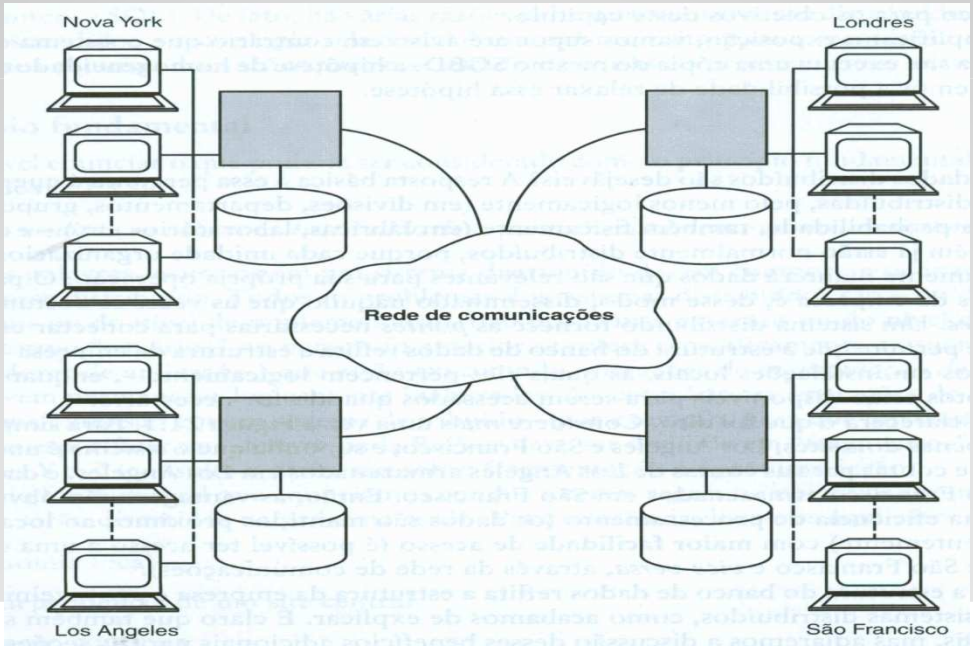
- Necessidade
- Benefícios
- Trabalho de Gonçalves (2007)
  - Driver JDBC
  - Acesso a Banco de Dados Distribuído (BDD)
  - Fragmentação Horizontal
  - Esquema Conceitual Global (ECG)
  - Restrições

Objetivos

- Disponibilizar soluções para as consultas sobre contexto global
  - Com agrupamento
  - Com ordenação
- Possibilitar o uso do `PreparedStatement` no *driver* JDBC
- Correções na Ferramenta de Edição de ECG

Fundamentação Teórica

- Banco de Dados Distribuído (BDD)
  - Formas de armazenamento
  - Transparência



Fonte: DATE (2000, p. 563).

Fundamentação Teórica

- Fragmentação Horizontal
  - Fragmento horizontal de uma relação

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

Fonte: SILBERSCHATZ (1999, p. 591).

  - Reconstrução da relação

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

Fonte: SILBERSCHATZ (1999, p. 591).

Fundamentação Teórica

- Agrupamento
  - Funções de agregação
    - MIN, MAX, COUNT, SUM, AVG
  - Cláusula GROUP BY
  - Restrições HAVING

```
SELECT      B.DS_TITULO,
            COUNT(A.CD_VOLUME)
FROM        GL_TITULO B,
            GL_VOLUME A
WHERE       A.CD_TITULO = B.CD_TITULO
GROUP BY    B.DS_TITULO
HAVING      COUNT(A.CD_VOLUME) > 6
```

Fundamentação Teórica

- Ordenação
  - Cláusula ORDER BY
  - Ascendente e descendente

```
SELECT      DS_TITULO,
            ANO_LANCAMENTO
FROM        GL_TITULO
ORDER BY    ANO_LANCAMENTO DESC
```

Fundamentação Teórica

- Driver JDBC
  - API Java
  - Independência de Banco de Dados
  - Um driver para cada BD
- Driver JDBC de Gonçalves (2007)
  - BDD
  - ECG
  - Funcionamento através de único driver
  - Restrições

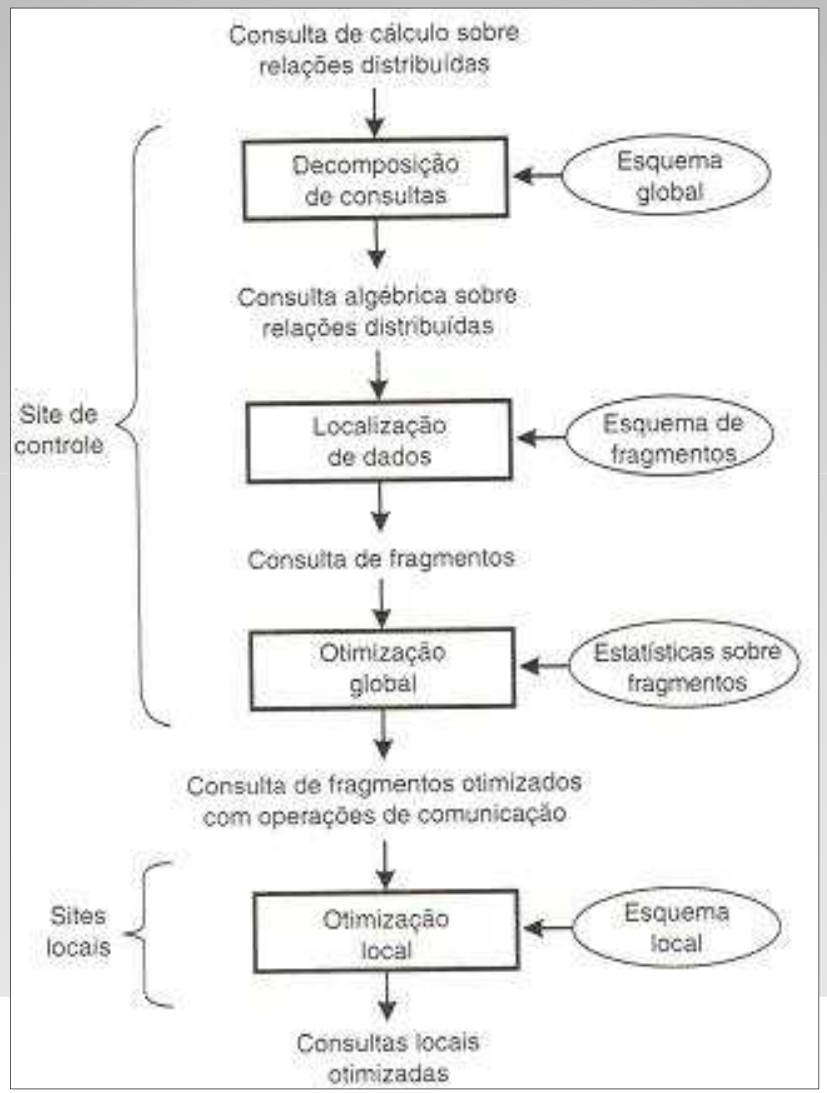
Fundamentação Teórica

- Processamento de consultas
  - Algoritmos de agrupamento e agregação
    - Número de passagens
    - Classificação, Hash
  - Algoritmos de ordenação
    - Memória, Quick-sort, Sort-Merge externo

Fundamentação Teórica

- Processamento de consultas em contexto distribuído
  - Alto nível global -> Baixo nível local
  - 4 camadas de processamento

Fundamentação Teórica



Trabalhos correlatos

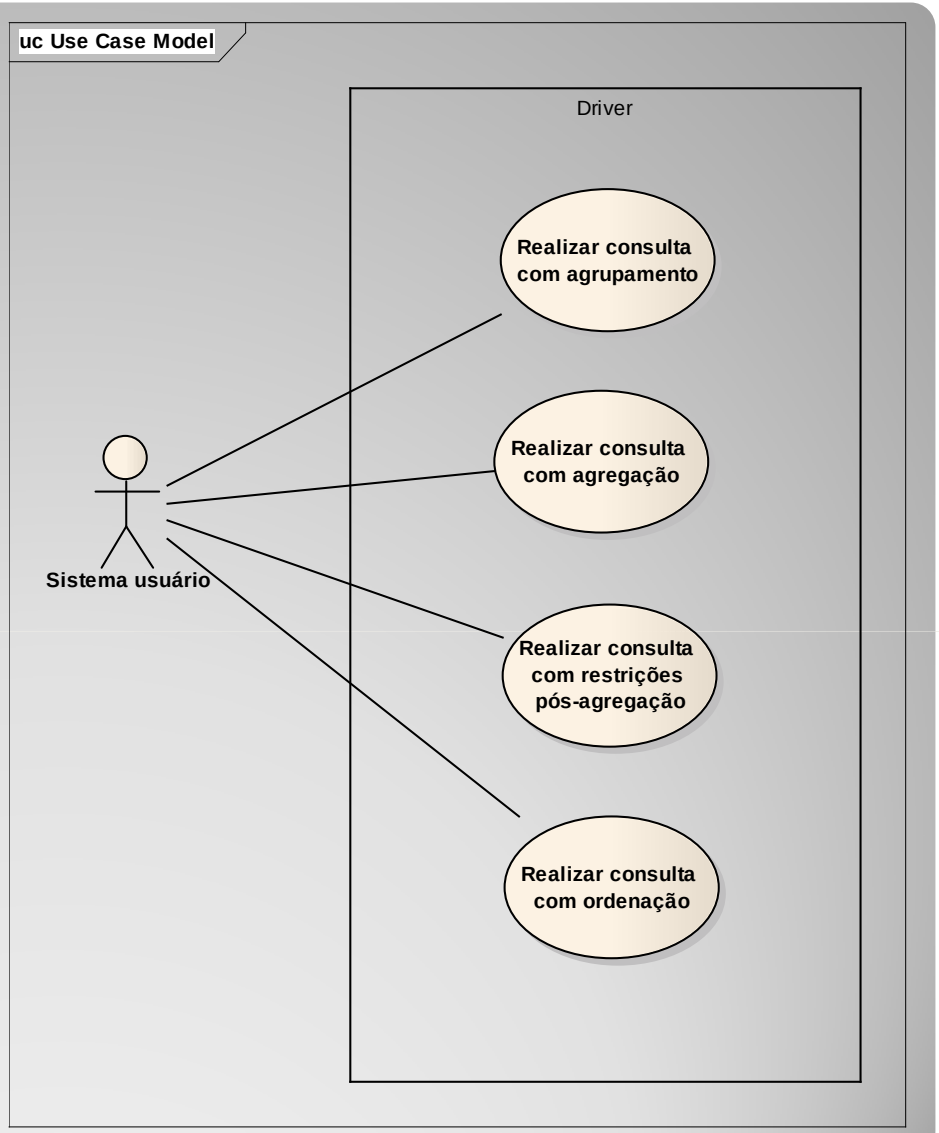
- Gianisini (2006)
  - Framework
  - Replicação local/remota
  - Troca de mensagens (JMS)
  - Hibernate
  - Java Transaction API (JTA)

Requisitos

- Consultas com GROUP BY
- Consultas com MIN, MAX, SUM, COUNT e AVG
- Consultas com ORDER BY
- Utilização de PreparedStatement nas consultas

Especificação

- Casos de Uso



Especificação

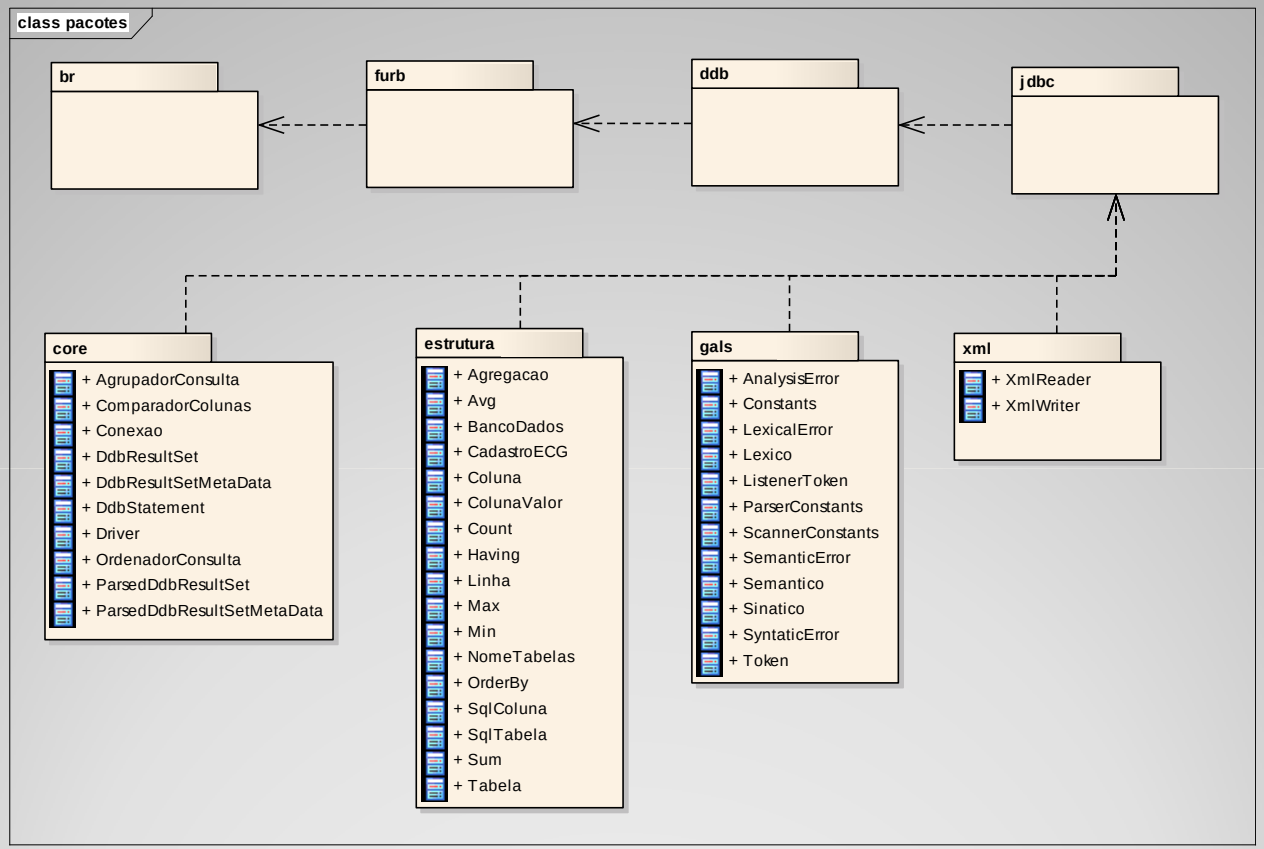
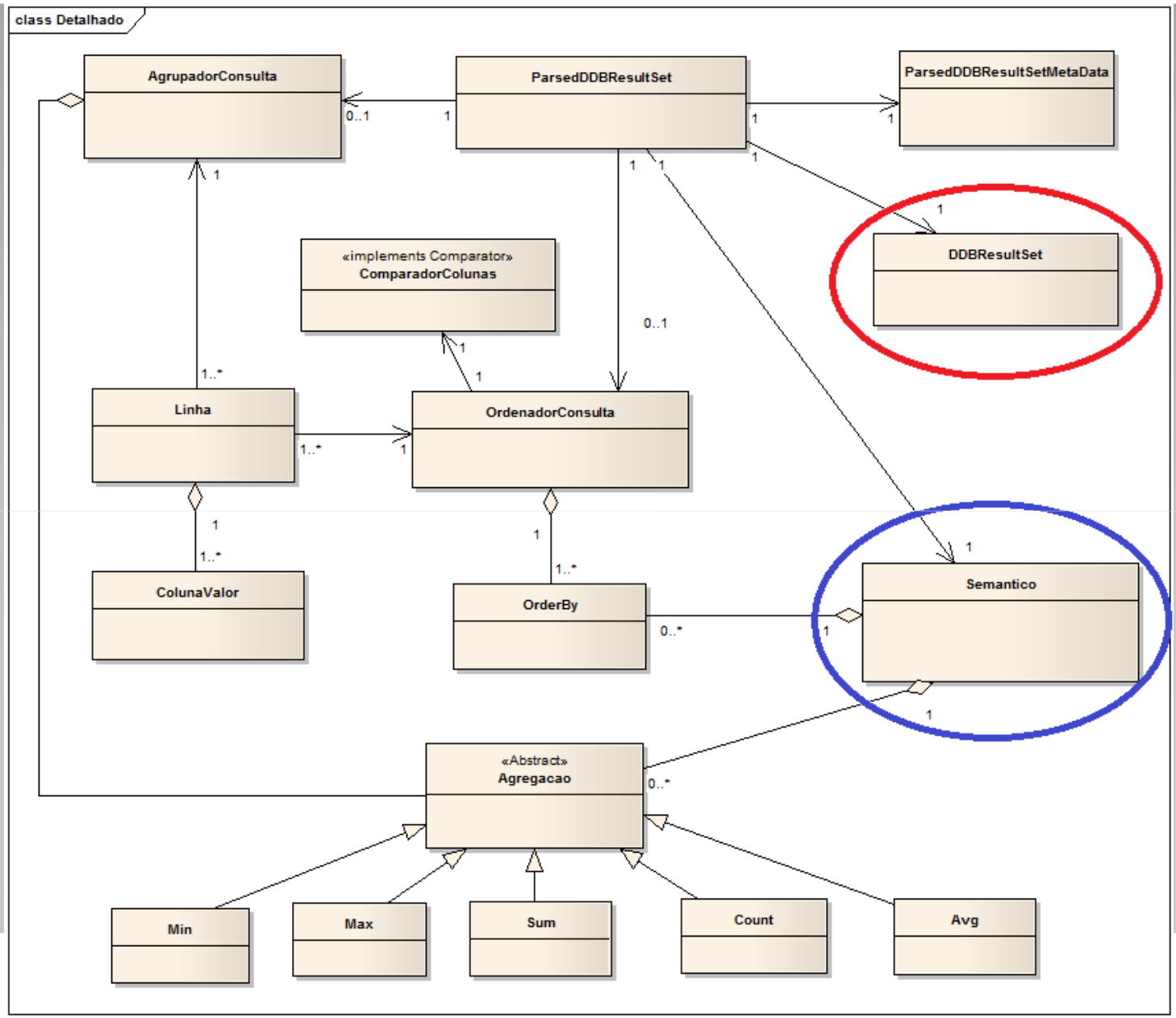
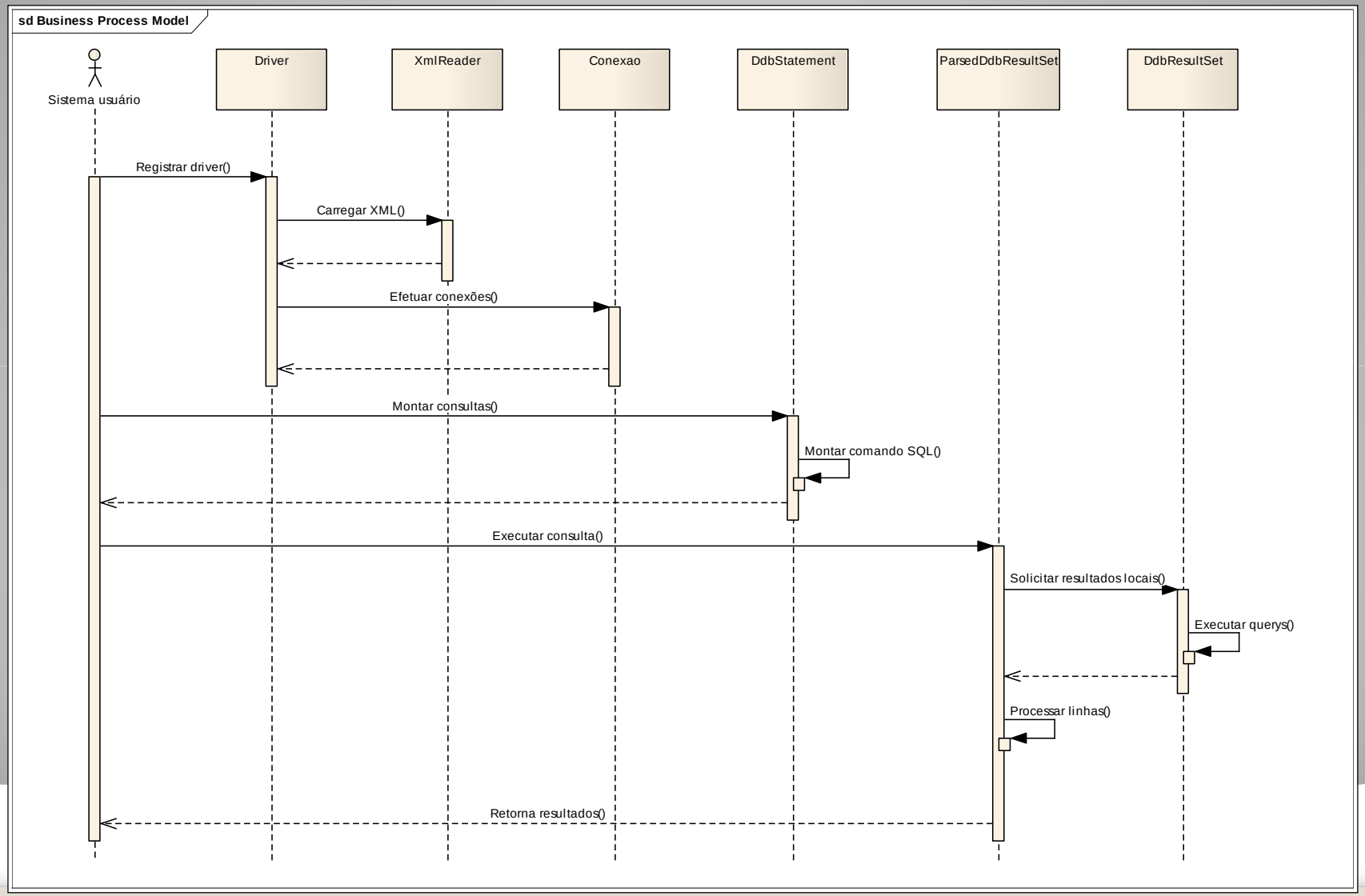


Diagrama de Pacotes


class Detalhado
AgrupadorConsulta
ParsedDDBResultSet
ParsedDDBResultSetMetaData
DDBResultSet
«implements Comparator»
ComparadorColunas
Linha
OrdenadorConsulta
ColunaValor
OrderBy
Semantico
«Abstract»
Agregacao
Min
Max
Sum
Count
Avg
0..1
1
1..*
0..*

Especificação



Desenvolvimento

```
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,agregacao, having);
22         linhas = ac.executar();
23      }
24      //Processar ordenação
25      ArrayList<OrderBy> listaOrderBy = semanticoConsulta.getOrderBy();
26      if (listaOrderBy.size() > 0){
27         OrdenadorConsulta oc = new OrdenadorConsulta(linhas,listaOrderBy);
28         linhas = oc.executar();
29      }
30      primeiroNext = false;
31   }
32   this.cursor++;
33   return this.cursor != linhas.size();
34}
```

Método `next()`da classe `ParsedDdbResultSet`

GROUP BY

```
01 public ArrayList<Linha> executar(){
02    HashMap linhasAgrupadas = new HashMap();
03
04    for (int i=0; i < linhas.size(); i ++){
05        Linha linha = linhas.get(i);
06        //Limpar valores do groupBy
07        List<Object> groupBy = new ArrayList<Object>();
08        //Verificar se a coluna está no group by
09        for (int j=0; j < colunasGroupBy.size(); j++){
10            String nome2 = colunasGroupBy.get(j);
11            for (int k=0; k < linha.getColunas().size(); k++){
12               String nome1 = linha.getColuna(k+1).getNome();
13               if (nome1.equalsIgnoreCase(nome2)){
14                   groupBy.add(linha.getColuna(k+1).getValor());
15               }
16            }
17        }
18        //Verificar se a linha já está inserida
19        List<Agregacao> agregacoesAtual = (List<Agregacao>) 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 }
```

Método `executar()` da classe `AgrupadorConsulta`

Agregações

```
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();
}
```

COUNT

```
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 = Integer.parseInt(linha.getColuna(super.getColuna()).getValor().toString());
        } else {
            atual =(Integer)atual +
                    Integer.parseInt(linha.getColuna(super.getColuna()).getValor().toString());
        }
    }
    @Override
    public Object clone() throws CloneNotSupportedException {
        return new Count(super.getColuna());

    }

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

SUM

```
public void executar(Linha linha) {
        if (atual == null){
            atual = linha.getColuna(super.getColuna()).getValor();
        }
        else if (linha.getColuna(super.getColuna()).getValor().getClass() == String.class){
            atual = 0;
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Integer.class){
            Integer valor1 = (Integer)atual;
            Integer valor2 = (Integer)linha.getColuna(super.getColuna()).getValor();
            atual = valor1 + valor2;
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Float.class){
            Float valor1 = (Float)atual;
            Float valor2 = (Float)linha.getColuna(super.getColuna()).getValor();
            atual = valor1 + valor2;
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Double.class){
            Double valor1 = (Double)atual;
            Double valor2 = (Double)linha.getColuna(super.getColuna()).getValor();
            atual = valor1 + valor2;
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Date.class){
            Date valor1 = (Date)atual;
            Date valor2 = (Date)linha.getColuna(super.getColuna()).getValor();
            atual = valor1.getDate() + valor2.getDate();
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Time.class){
            Time valor1 = (Time)atual;
            Time valor2 = (Time)linha.getColuna(super.getColuna()).getValor();
            atual = valor1.getTime() + valor2.getTime();
        }
}
```

Método `executar()` da classe `Sum`

MIN e MAX

```
public void executar(Linha linha) {
        if (atual == null){
            atual = linha.getColuna(super.getColuna()).getValor();
        }
        else if (linha.getColuna(super.getColuna()).getValor().getClass() == String.class){
            String valor1 = (String)atual;
            String valor2 = (String)linha.getColuna(super.getColuna()).getValor();
            if (valor1.compareTo(valor2) > 0){
                atual = (String)linha.getColuna(super.getColuna()).getValor();
            }
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Integer.class){
            Integer valor1 = (Integer)atual;
            Integer valor2 = (Integer)linha.getColuna(super.getColuna()).getValor();
            if (valor1.compareTo(valor2) > 0){
                atual = (Integer)linha.getColuna(super.getColuna()).getValor();
            }
...
        } else if (linha.getColuna(super.getColuna()).getValor().getClass() == Date.class){
            Date valor1 = (Date)atual;
            Date valor2 = (Date)linha.getColuna(super.getColuna()).getValor();
            if (valor1.compareTo(valor2) > 0){
                atual = (Date)linha.getColuna(super.getColuna()).getValor();
            }
...
}
```

Método `executar()`da classe `Min`

AVG

```
public void executar(Linha linha) {
     if (soma == null){
         soma = Double.parseDouble(linha.getColuna(super.getColuna()).getValor().toString());
     } else if ((linha.getColuna(super.getColuna()).getValor().getClass() == Integer.class) ||
                 (linha.getColuna(super.getColuna()).getValor().getClass() == Long.class) ||
                 (linha.getColuna(super.getColuna()).getValor().getClass() == Float.class) ||
                 (linha.getColuna(super.getColuna()).getValor().getClass() == Double.class) ||
                 (linha.getColuna(super.getColuna()).getValor().getClass() == BigDecimal.class)){
         soma = (Double)soma +
         Double.parseDouble(linha.getColuna(super.getColuna()).getValor().toString());
     } else {
         soma = (Double)0.00;
     }
     qtd++;
}

public Object getAtual() {
     if (qtd > 0){
         return (Double)soma/qtd;
     } else {
         return null;
     }
}
```

Métodos `executar()` e `getAtual()` da classe AVG

ORDER BY

```
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;
    }
}
```

ORDER BY

```
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 colunas da linha e verificar se são 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;
}
```

Método `compare` da classe `ComparadorColunas`

HAVING

```
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().equalsIgnoreCase(ag.getTipo())) &&
08           (restricoesHaving.get(j).getAgregacao().getColuna().equalsIgnoreCase(ag.getColuna()))){
09            //Verificar restricao
10            if ((ag.getAtual().getClass() == Integer.class) ||
11                (ag.getAtual().getClass() == Long.class) ||
12                (ag.getAtual().getClass() == Float.class) ||
13                (ag.getAtual().getClass() == Double.class) ||
14                (ag.getAtual().getClass() == BigDecimal.class)){
15                Double agregado = Double.parseDouble(ag.getAtual().toString());
16                Double restricao = Double.parseDouble(
17                                restricoesHaving.get(j).getValorRestricao().toString());
18                if (restricoesHaving.get(j).getRestricao().equals("=")){
19                    if (agregado.compareTo(restricao) == 0){
20                        incluiLinha = true;
21                    }
22 ...
```

Método `verificaHaving` da classe `AgrupadorConsulta`

Considerações finais

- Desempenho
- Validação
- Restrições

	Gonçalves/Roeder	Gianisini
Foco	BDD	BDD
Permite	Consultas	Replicação
Tecnologia	JDBC	Hibernate
BD Heterogêneos	Sim	Sim

Conclusão

- Objetivos alcançados
  - Consultas dist. com agrup./ordenação
  - Correções na Ferramenta de Edição de ECG
  - Interface PreparedStatement
- Relevância do trabalho
- Continuidade

Conclusão

- Extensões
  - Comandos DML
  - Trat. falhas de conexão e comunicação
  - JOIN nas consultas globais
  - Select específico de um BD ser transformado
  - Apelidos para as colunas da ordenação
  - Fórmulas em agreg. e agreg. composta
  - Eliminação de duplicatas
  - Situações extremas, memória
  - Otimização de consultas
  - Chaves primárias para as tabelas globais
  - Tratamento de mens. erro nas consultas

Obrigado!