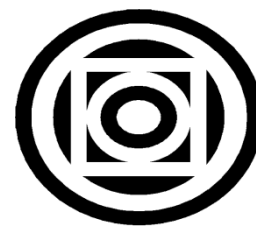




UNIVERSIDADE FEDERAL DE MATO GROSSO
CAMPUS UNIVERSITÁRIO DO ARAGUAIA
INSTITUTO DE CIÊNCIAS EXATAS E DA TERRA
CIÊNCIA DA COMPUTAÇÃO



Viveiro do Vale
Trabalho de Banco de Dados

Walas Jhony

Barra do Garças

2013

Sumário

1. Introdução.....	6
2. Modelo Entidade Relacionamento – MER.....	7
3. Mapeamento	8
3.1. MUDAS	8
3.2. MUDAS PLANTADAS ¹	8
3.3. SÓCIO	8
3.4. SOCIO_TRABALHOU	8
3.5. ADUBOS	8
3.6. ADUBOS_GASTOS	8
3.7. VENDA.....	8
3.8. VENDA_INCLUI	8
Detalhes da base de dados.....	9
4. SQL de criação da base	10
4.1. Tabela “Mudas”.....	10
4.2. Tabela “Mudas Plantadas”	10
4.3. Tabela “Sócio”	10
4.4. Tabela “Sócio Trabalhou”	10
4.5. Tabela “Adubo”.....	11
4.6. Tabela “Adubo Gasto”	11
4.7. Tabela “Venda”	11
4.8. Tabela “Venda Inclui”	11
4.9. Trigger “Atualiza Muda Plantio	12
4.10. Trigger “Atualiza Muda Venda”	12
4.11. Trigger “Atualiza Adubo”	12
5. Telas do sistema e SQL associado	13
5.1. Tela principal do programa	13

5.2. Tela de cadastro e atualização de novos adubos	13
5.3. Tela de listagem de adubos	14
5.4. Tela de cadastro de um novo sócio	14
5.5. Tela de listagem de sócios.....	15
5.6. Tela de cadastro de um novo plantio.....	15
5.7. Tela de busca de espécies	16
5.8. Tela de venda	17
5.9. Telas de relatório	18
5.9.1. Relatório sobre adubos	18
5.9.2. Relatório sobre mudas	19
5.9.3. Relatório sobre sócios	20
5.9.4. Relatório sobre vendas	21
Interações com a base de dados	22
6.1. Classe Conexão	22
6.2. Classe AduboDAO	22
6.2.1. Método <i>inserir</i>	22
6.2.2. Método <i>realizaBusca</i>	23
6.2.3. Método <i>busca</i> (sem parâmetro).....	23
6.2.4. Método <i>busca</i> (com parâmetro)	24
6.2.5. Método <i>buscaPreco</i>	24
6.3. Classe AduboGastoDAO.....	25
6.3.1. Método <i>insere</i>	25
6.4. Classe MudasDAO	25
6.4.1. Método <i>insere</i>	25
6.4.2. Método <i>realizaBusca</i>	26
6.4.3. Método <i>busca</i> (sem parâmetro).....	26
6.4.4. Método <i>busca</i> (com parâmetro)	26

6.5. Classe MudasPlantadosDAO	27
6.5.1. Método <i>insere</i>	27
6.5.2. Método <i>buscaTotal</i>	28
6.6. Classe SócioDAO.....	29
6.6.1. Método <i>insere</i>	29
6.6.2. Método <i>realizaBusca</i>	30
6.6.3. Método <i>buscaAll</i>	30
6.6.4. Método <i>buscaPorCpf</i>	31
6.7. Classe SócioTrabalhouDAO	31
6.7.1. Método <i>insere</i>	31
6.8. Classe VendaDAO	32
6.8.1. Método <i>insere</i>	32
6.8.2. Método <i>proxNumero</i>	32
6.9. Classe VendaIncluiDAO	33
6.9.1. Método <i>inseri</i>	33
6.10. Classe RelatórioAdubo.....	33
6.10.1. Método <i>relatorioAdubo</i>	33
6.10.2. Método <i>relatorioAduboEspecie</i>	34
6.10.3. Método <i>relatorioAduboData</i>	34
6.10.4. Método <i>relatorioAduboGasto</i>	34
6.10.5. Método <i>relatorioAduboAllUntilToday</i>	34
6.11. Classe RelatórioMudas.....	35
6.11.1. Método <i>relatorioMudas</i>	35
6.11.2. Método <i>relatorioMudasEspecie</i>	35
6.11.3. Método <i>relatorioMudasData</i>	35
6.11.4. Método <i>relatorioMudasSocio</i>	36
6.11.5. Método <i>relatorioMudasAllUntilToday</i>	36

6.12. Classe RelatórioSócio	36
6.12.1. Método <i>relatorioSocioEspecie</i>	36
6.12.2. Método <i>relatorioSocioData</i>	37
6.12.3. Método <i>relatorioSocioHorasData</i>	38
6.12.4. Método <i>relatorioSocioTotalHoras</i>	39
6.13. Classe RelatórioVenda	40
6.13.1. Método <i>relatorioVenda</i>	40
6.13.2. Método <i>relatorioVendaMais</i>	40
6.13.3. Método <i>relatorioVendaQuant</i>	41
6.13.4. Método <i>relatorioVendaData</i>	41
6.13.5. Método <i>relatorioVendaAll</i>	41

1. Introdução

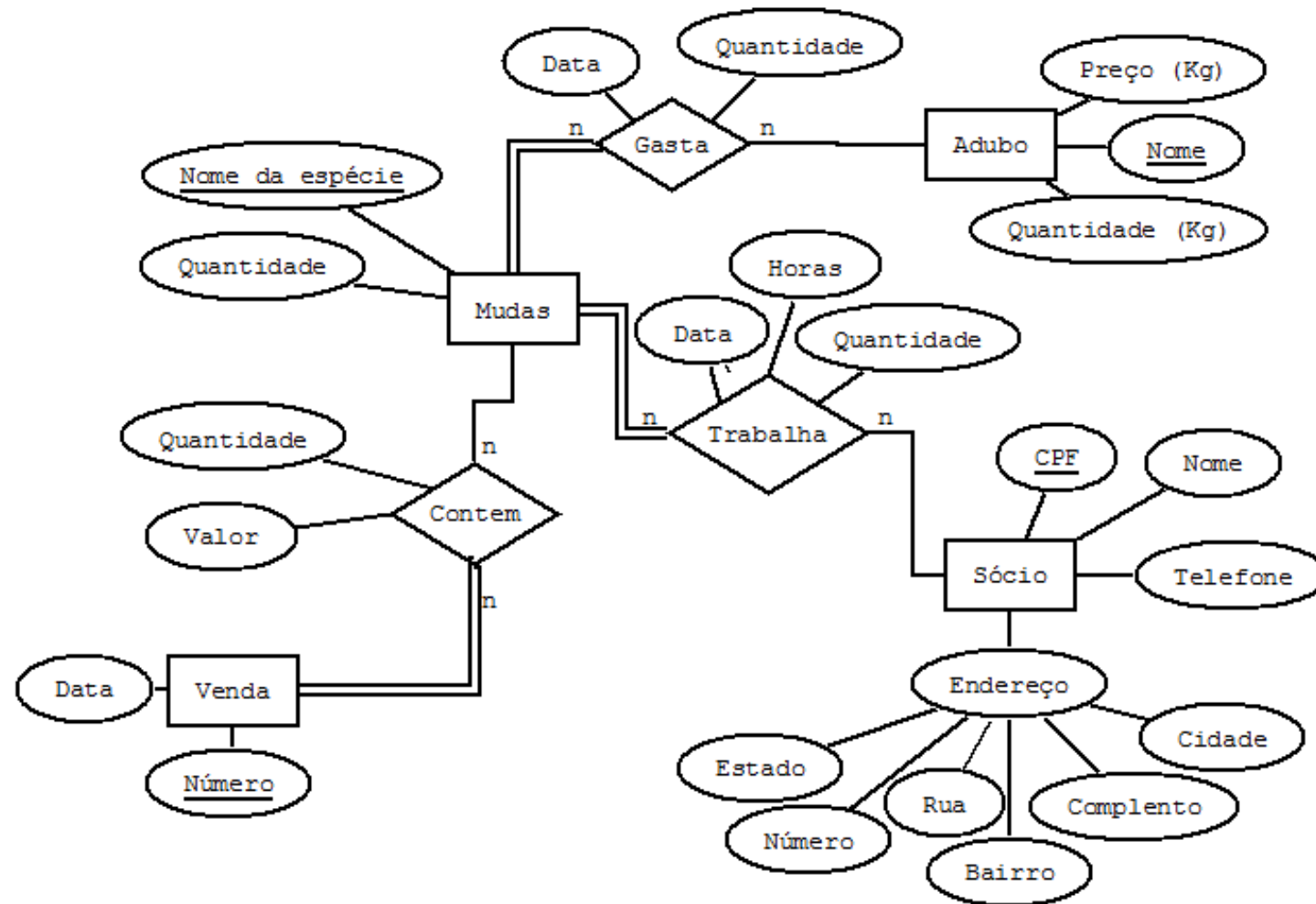
Este trabalho tem como objetivo testar os conhecimentos referentes à disciplina Banco de Dados. Para tal, deve ser desenvolvido um sistema qualquer que utilize um banco de dados para armazenar as informações do sistema.

A proposta para deste trabalho é desenvolver um sistema para um viveiro que vende mudas de plantas nativas do cerrado. Este viveiro trabalha em forma de sociedade, onde cada sócio trabalha em prol do viveiro. Todos os sócios são responsáveis pelas vendas, não havendo comissão por vendas realizadas.

O sistema deve armazenar dados referentes às mudas disponíveis para venda, dados referentes aos adubos utilizados no plantio das mudas e dados referentes às horas trabalhadas de cada sócio. O sistema deve ser também capaz de gerar relatórios referentes aos adubos gastos, às mudas plantadas, às horas trabalhadas e às vendas. Sobre as mudas deseja-se guardar o nome de cada espécie, a quantidade de mudas disponíveis para venda, o dia do plantio. Sobre os adubos deseja-se guardar o nome de cada adubo e a quantidade disponível para o plantio das mudas. O sistema deve informar ser capaz de gerar vários tipos de relatório para cada informação (adubo, mudas, sócio, vendas).

2. Modelo Entidade Relacionamento – MER

A partir dos requisitos levantados, foi extraído o seguinte Modelo Entidade Relacionamento.



3. Mapeamento

3.1. **MUDAS**

(nome, quantidade);

Chave primaria: nome.

3.2. **MUDAS PLANTADAS**¹

(especie, data, quantidade);

Chave primaria: nome, data.

Chave estrangeira:

especie faz referencia à tabela MUDAS

3.3. **SÓCIO**

(cpf, nome, rua, numero, bairro, complemento, cidade, estado, telefone);

Chave primaria: cpf.

3.4. **SOCIO TRABALHO**

(cpf_fk, especie_fk, data, horas);

Chave primaria: cpf_fk, especie_fk, data.

Chave estrangeira:

cpf_fk faz referencia à tabela SOCIO.

especie_fk faz referencia à tabela MUDAS.

3.5. **ADUBOS**

(nome, quantidade, preço);

Chave primaria: nome.

3.6. **ADUBOS GASTOS**

(especie_fk, nome_ad_fk, quant_ad, data);

Chave primaria: especie_fk, nome_ad_fk, data.

Chave estrangeira:

especie_fk faz referencia à tabela MUDAS.

nome_ad_fk faz referencia à tabela ADUBO.

3.7. **VENDA**

(numero, data);

Chave primaria: numero.

3.8. **VENDA INCLUI**

(num_venda, especie_fk, quantidade, valor);

Chave primaria: numero_fk, especie_fk.

Chave estrangeira:

especie_fk faz referencia à tabela MUDAS.

num_venda faz referencia à tabela VENDA.

Detalhes da base de dados

A partir do Modelo Entidade Relacionamento, foi feito o mapeamento para criação da base de dados. Para este trabalho foi utilizado o sistema gerenciador de banco de dados MySQL, versão 5.6.

Sobre as tabelas temos algumas informações:

- A tabela “MUDAS” contém o nome da espécie e a quantidade de mudas disponível para venda. Já a tabela “MUDAS_PLANTADAS” contém informações como o nome da espécie, a data e a quantidade de mudas plantadas nesta data.
- A tabela “SÓCIO” contém as informações pessoais de cada sócio. Já a tabela “SÓCIO_TRABALHOU” contém data em que cada sócio trabalhou, a espécie e as horas trabalhadas nesta data.
- A tabela “ADUBO” contém o nome, a quantidade disponível e o último preço pago por quilograma de cada tipo de adubo. Já a tabela “ADUBO_GASTO” contém a data do plantio, nome da espécie plantada e a quantidade de cada adubo utilizado na plantação desta espécie nesta data.
- A tabela “VENDA” contém o número e a data de cada venda realizada. Já a tabela “VENDA_INCLUI” contém o nome das espécies incluídas em cada venda, bem como o preço por muda.

¹ - A tabela “MUDAS_PLANTADAS” surgiu da normalização da base de dados, pois a quantidade não tem dependência total com a chave pelo motivo de depender somente da espécie plantada, o que infringe a segunda Forma Normal.

4. SQL de criação da base

Segue o código MySQL para criação da base de dados:

4.1. Tabela “Mudas”

```
CREATE TABLE MUDAS (  
    nome VARCHAR(50),  
    quantidade INT NOT NULL,  
    PRIMARY KEY (nome)  
);
```

4.2. Tabela “Mudas Plantadas”

```
CREATE TABLE MUDAS_PLANTADAS (  
    especie VARCHAR(50),  
    data DATE,  
    quant INT,  
    PRIMARY KEY (especie, data, quant),  
    FOREIGN KEY (especie) REFERENCES MUDAS (nome)  
        ON DELETE CASCADE  
);
```

4.3. Tabela “Sócio”

```
CREATE TABLE SOCIO (  
    cpf VARCHAR (14),  
    nome VARCHAR (80) NOT NULL,  
    rua VARCHAR (80),  
    numero INT DEFAULT 0,  
    bairro VARCHAR (50),  
    complemento VARCHAR (80),  
    cidade VARCHAR (50) NOT NULL,  
    estado INT NOT NULL,  
    telefone VARCHAR (14),  
    PRIMARY KEY (cpf)  
);
```

4.4. Tabela “Sócio Trabalhou”

```
CREATE TABLE SOCIO_TRABALHOU (  
    cpf_fk VARCHAR(14),  
    especie_fk VARCHAR(50),  
    data DATE,  
    hora TIME,  
    PRIMARY KEY (cpf_fk , especie_fk, data),  
    FOREIGN KEY (cpf_fk) REFERENCES SOCIO (cpf)  
        ON DELETE CASCADE,  
    FOREIGN KEY (especie_fk) REFERENCES MUDAS (nome)  
        ON DELETE CASCADE  
);
```

4.5. Tabela “Adubo”

```
CREATE TABLE ADUBO (  
    nome VARCHAR(50),  
    quantidade FLOAT NOT NULL,  
    preco FLOAT NOT NULL,  
    PRIMARY KEY (nome)  
);
```

4.6. Tabela “Adubo Gasto”

```
CREATE TABLE ADUBO_GASTO (  
    especie_fk VARCHAR(50),  
    nome_ad_fk VARCHAR(50),  
    quant_ad FLOAT NOT NULL,  
    data DATE,  
    PRIMARY KEY (especie_fk , nome_ad_fk, data),  
    FOREIGN KEY (especie_fk) REFERENCES MUDAS (nome)  
        ON DELETE CASCADE,  
    FOREIGN KEY (nome_ad_fk) REFERENCES ADUBO (nome)  
        ON DELETE CASCADE  
);
```

4.7. Tabela “Venda”

```
CREATE TABLE VENDA (  
    numero INT AUTO_INCREMENT,  
    data DATE NOT NULL,  
    PRIMARY KEY (numero)  
);
```

4.8. Tabela “Venda Inclui”

```
CREATE TABLE VENDA_INCLUI (  
    num_venda INT,  
    especie VARCHAR(50),  
    quantidade INT NOT NULL,  
    preco FLOAT NOT NULL,  
    PRIMARY KEY (num_venda , especie),  
    FOREIGN KEY (num_venda) REFERENCES VENDA (numero)  
        ON DELETE CASCADE,  
    FOREIGN KEY (especie) REFERENCES MUDAS (nome)  
        ON DELETE CASCADE  
);
```

4.9. Trigger “Atualiza Muda Plantio

```
CREATE TRIGGER atualizaMudasPlantio
after insert on mudas_plantadas
for each row
begin
    update mudas
    set quantidade = quantidade + new.quant
    where nome = new.especie;
end;
```

4.10. Trigger “Atualiza Muda Venda”

```
CREATE TRIGGER atualizaMudasVenda
after insert on venda_inclui
for each row
begin
    update mudas
    set quantidade = quantidade - new.quantidade
    where nome = new.especie;
end;
```

4.11. Trigger “Atualiza Adubo”

```
CREATE TRIGGER atualizaAdubo
after insert on adubo_gasto
for each row
begin
    update adubo
    set quantidade = quantidade - new.quant_ad
    where nome = new.nome_ad_fk;
end;
```

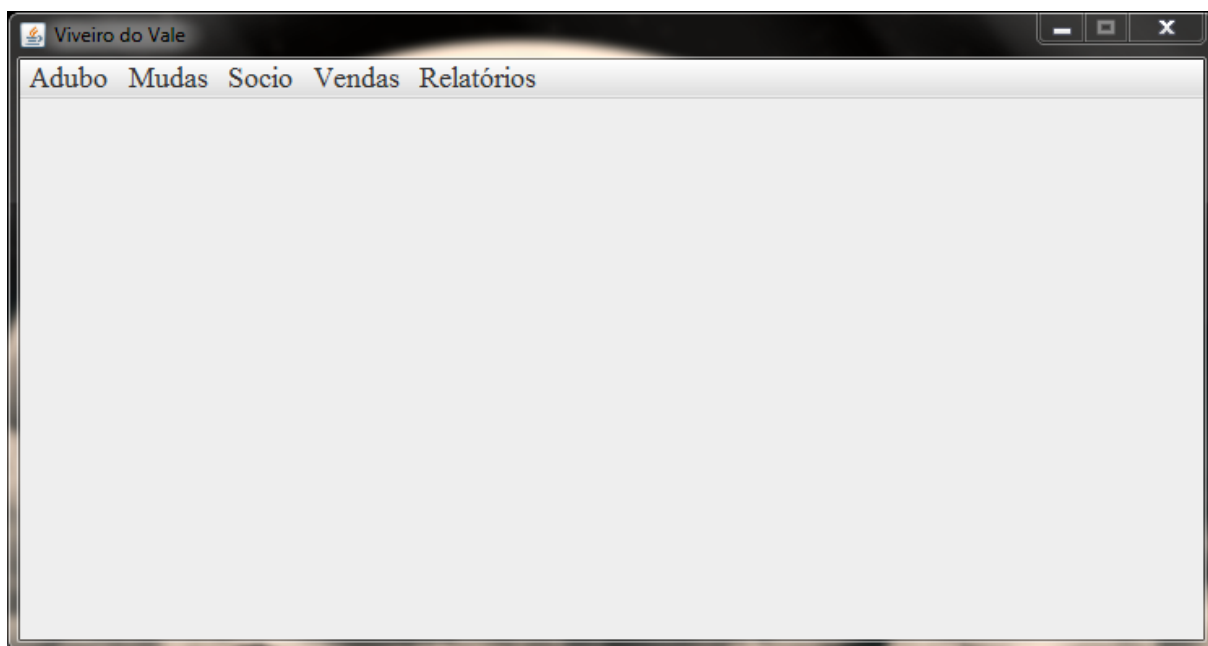
O trigger “Atualiza Mudas Plantio” foi criado para manter a consistência na base de dados quando é realiza um venda, já que a quantidade de mudas disponíveis para venda aumenta.

O trigger “Atualiza Mudas Venda” foi criado para manter a consistência na base de dados quando é realiza uma venda, já que a quantidade de mudas disponíveis para venda diminui.

O trigger “Atualiza Adubo” foi criado para manter a consistência na base de dados quando há um novo plantio, já que são gastos alguns adubos o que diminui a quantidade disponível.

5. Telas do sistema e SQL associado

5.1. Tela principal do programa




5.2. Tela de cadastro e atualização de novos adubos

A imagem mostra a janela de cadastro e atualização de novos adubos. No topo, há uma barra de menu com as opções 'Adubo', 'Mudas', 'Socio', 'Vendas' e 'Relatórios'. Abaixo, há três campos de entrada de texto rotulados 'Nome:', 'Quantidade (em kg):' e 'Preço (por kg):'. Abaixo dos campos, há três botões: 'Salvar', 'Limpar' e 'Cancelar'.

Nesta tela é feito o cadastro de um novo adubo. Caso ele já exista na base de dados, sua quantidade é atualizada e passa a ser a quantidade já existente mais a nova quantidade inserida e o preço passa a ser o novo preço inserido. Nesta tela é chamado método [*insere*](#) da classe AduboDAO. Após a interação com a base de dados é exibida uma mensagem ao usuário informando-o se o adubo foi inserido ou atualiza, ou se houve algum erro.

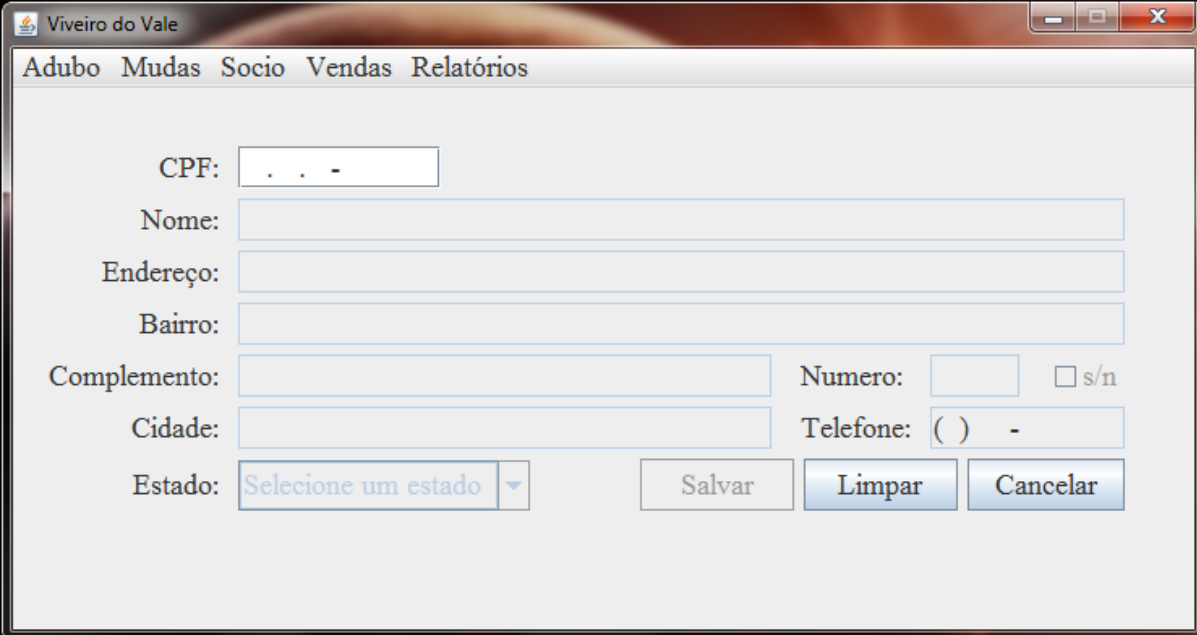
5.3. Tela de listagem de adubos



Nome	Quantidade	Preço por Kg
Amônia	90.8	9.0
Calcário	86.85	5.3
Cálcio	86.445	5.5
Fosfato de Sódio	90.125	8.9
Super Simples	94.8	4.5

Nesta tela são mostrados todos os adubos cadastrados na base de dados, sua quantidade disponível e o seu preço. Antes da abertura do painel, é chamado o método [busca](#) da classe AduboDAO, que retorna a lista dos adubos mostrados no painel. Caso não haja adubos cadastrados na base o usuário é informado através de uma mensagem.

5.4. Tela de cadastro de um novo sócio



CPF:

Nome:

Endereço:

Bairro:

Complemento: Numero: ☐ s/n

Cidade: Telefone: () -

Estado:

Nesta tela é realizado o cadastro de um novo sócio. Assim que o usuário informa o CPF, e o mesmo é verificado, é realizada uma busca na base de dados através do método [buscaPorCpf](#). Caso o CPF esteja cadastrado na base, os campos são preenchidos com os

dados da base e caso o usuário queira fazer alguma alteração basta clicar em salvar. Caso o CPF não esteja cadastrado, os campos são abertos para edição.

5.5. Tela de listagem de sócios

The screenshot shows a window titled 'Viveiro do Vale' with a menu bar containing 'Adubo', 'Mudas', 'Socio', 'Vendas', and 'Relatórios'. The main area is titled 'Lista de socios' and contains a table with three columns: 'CPF', 'Nome', and 'Telefone'. The table is currently empty.

Do mesmo modo que na tela de listagem de adubo, este painel mostra a lista de todos sócios cadastrado na base de dados. Para isso, o método [buscaAll](#) é chamado antes da abertura do painel. Caso não haja sócios cadastrados na base, um alerta é mostrado ao usuário.

5.6. Tela de cadastro de um novo plantio

The screenshot shows a window titled 'Viveiro do Vale' with a menu bar containing 'Adubo', 'Mudas', 'Socio', 'Vendas', and 'Relatórios'. The window is divided into two main sections. The left section is titled 'Adubo / Quantidade' and contains three input fields: 'Data', 'Espécie', and 'Quantidade:'. There are two buttons: 'Limpar' next to the 'Data' field and 'Ok' next to the 'Quantidade:' field. The right section is titled 'Socio / Horas trabalhadas' and contains a large empty text area. At the bottom of the right section, there are two buttons: '- Socio' and '+ Socio'.

Nesta tela é realizado o cadastro de novos plantios. Para que isso aconteça é necessário que exista pelo menos um adubo e um sócio cadastrado na base e deve ser utilizado pelo menos um adubo e pelo menos um sócio deve ter trabalhado neste plantio. Caso umas dessas condições sejam quebradas, um alerta é mostrado ao usuário.

Quando o usuário clica em “+ Adubo” é mostrada uma relação com a quantidade de adubos disponíveis. Do mesmo modo, é mostrada uma relação de todos sócios quando o usuário clica em “+ Sócio”.

The image shows two separate application windows. The left window has a title bar with a close button (X) and contains a table with two columns: 'Nome' and 'Quantidade'. It lists five items, each with a checkbox and a text input field for the quantity.

Nome	Quantidade
<input type="checkbox"/> Amônia	90.8
<input type="checkbox"/> Calcário	86.85
<input type="checkbox"/> Cálcio	86.445
<input type="checkbox"/> Fosfato de Sódio	90.125
<input type="checkbox"/> Super Simples	94.8

The right window also has a title bar with a close button (X) and contains a list of names, each preceded by a checkbox.

Nome
<input type="checkbox"/> Ana Lopes de Almeida
<input type="checkbox"/> David Mota
<input type="checkbox"/> Jose Renato
<input type="checkbox"/> Lorayne Pinheiro
<input type="checkbox"/> Mathias Fassini Mantelli
<input type="checkbox"/> Walas Jhony Lopes Moraes

Portanto para exibição deste painel, são chamados os métodos [busca](#) e [buscaAll](#) das classe AduboDAO e SócioDAO, respectivamente.

Para finalizar o processo e salvar os dados na base, são chamados os métodos de inserção das classes [MudasDAO](#), [MudasPlantadasDAO](#), [AduboGastoDAO](#) e [SócioTrabalhouDAO](#).

5.7. Tela de busca de espécies

The image shows a screenshot of the 'Viveiro do Vale' application window. It has a title bar with standard window controls. Below the title bar is a menu bar with the options: 'Adubo', 'Mudas', 'Socio', 'Vendas', and 'Relatórios'. Below the menu bar is a search interface with a label 'Nome:' followed by a text input field. To the right of the input field are four buttons: 'Todas', 'Buscar', 'Limpar', and 'Cancelar'. Below the search interface is a table displaying search results.

arueira	900
ipe amarelo	800
ipe rosa	900
ipe roxo	600
pequi	1050

Nesta tela o usuário consegue fazer busca através do nome da espécie ou por parte do nome da espécie. O usuário consegue também visualizar todas as espécies cadastradas na base de dados bem como a quantidade disponível para venda.

Para realizara essas duas operações, são chamados os métodos [busca](#) (um sem parâmetro) e [busca](#) (com parâmetro).

5.8. Tela de venda

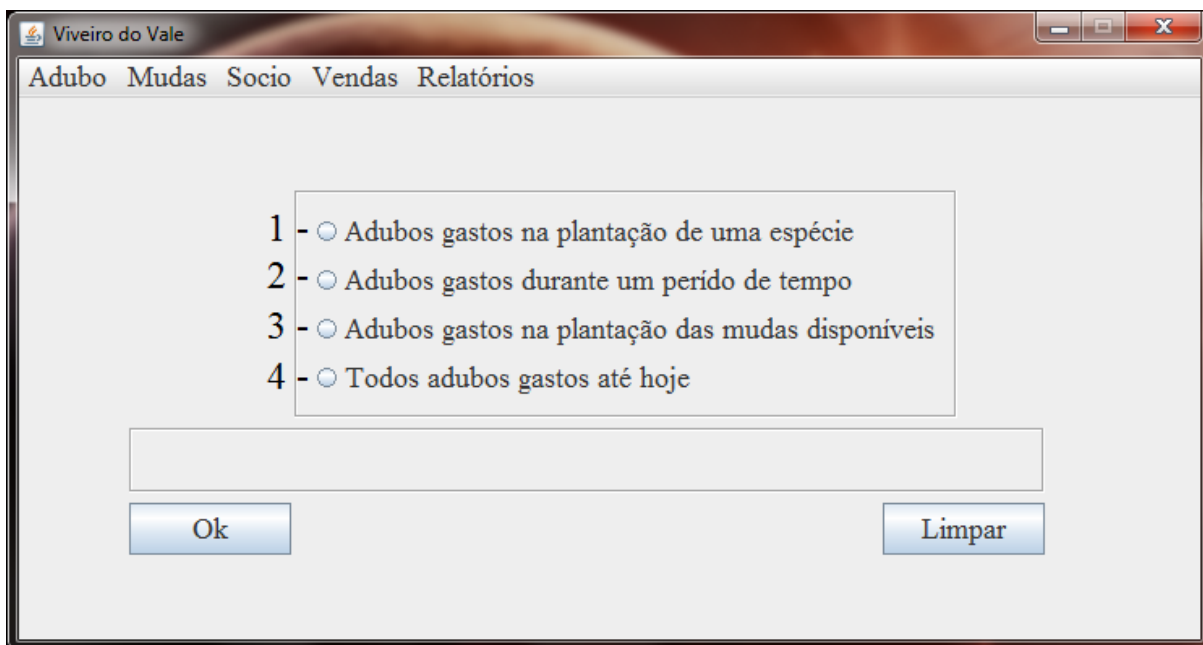
A imagem mostra a janela de aplicação 'Viveiro do Vale' com a aba 'Vendas' selecionada. O formulário é dividido em duas seções principais. À esquerda, há campos para 'Data:' e 'Valor:', cada um com um campo de entrada de texto correspondente. Abaixo desses campos estão os botões 'Ok' e 'Limpar'. À direita, há uma tabela com três cabeçalhos: 'Espécie:', 'Quantidade:' e 'Preço:'. O corpo da tabela é uma área vazia para a listagem de itens. Na base da interface, há dois botões adicionais: 'Remover' e 'Adicionar'.

Nesta é realizada um venda. Para isso é necessário adicionar pelo menos uma espécie de muda. Assim que isso é feito, será solicitado que o usuário informe a quantidade de mudas vendidas. Depois disso o sistema calcula o valor daquela quantidade de mudas e se for adicionado uma nova espécie, o sistema atualiza o valor da venda. Para persistir esses dados na base, são chamados os métodos [insere](#) da classe [VendaDAO](#) e [insere](#) da classe [VendaIncluiDAO](#).

5.9. Telas de relatório

Para cada categoria de relatório, temos quatro tipos, no cada um chama um método da sua classe de relatório.

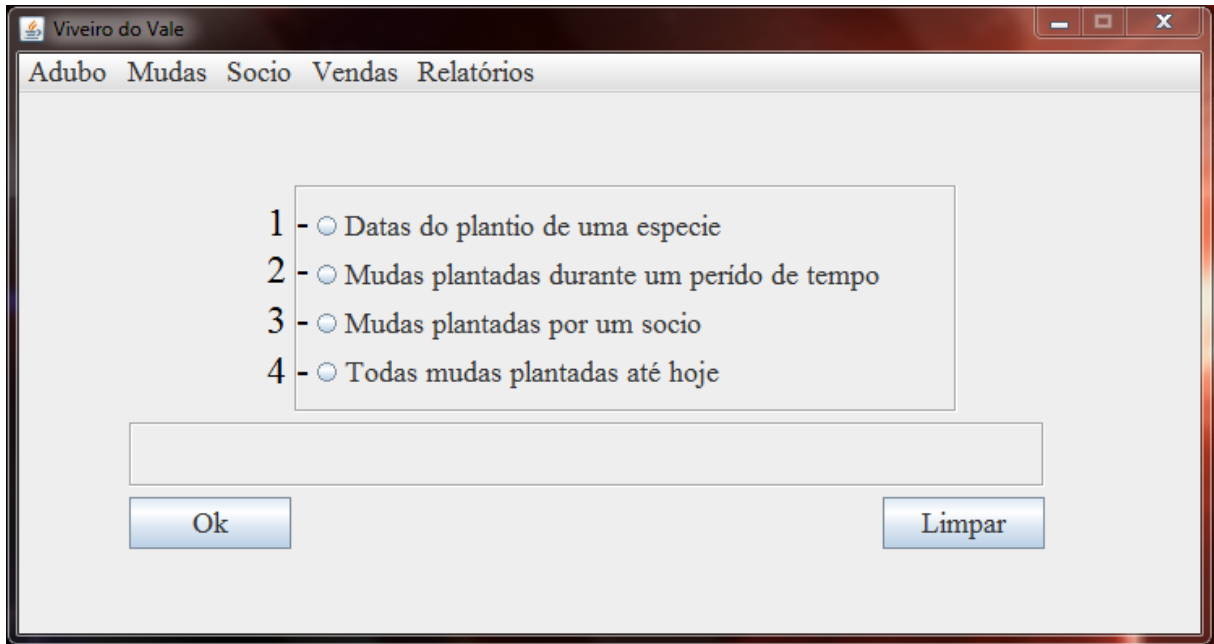
5.9.1. Relatório sobre adubos



Para estes relatórios com relação aos adubos temos a classe `RelatorioAdubo`, no qual cada um chama seu método:

1. Método chamado: [*relatorioAduboEspecie*](#).
 - Mostra todos os adubos utilizados na plantação de uma determinada espécie.
2. Método chamado: [*relatorioAduboData*](#).
 - Mostra todos os adubos e em quais espécies foram utilizados nos plantios realizados em um determinado período de tempo.
3. Método chamado: [*relatorioAduboGasto*](#).
 - Mostra todos os adubos gasto na produção das mudas que estão disponíveis para venda.
4. Método [*relatorioAduboAllUntilToday*](#).
 - Mostra todos os adubos gasto ate o momento a busca.

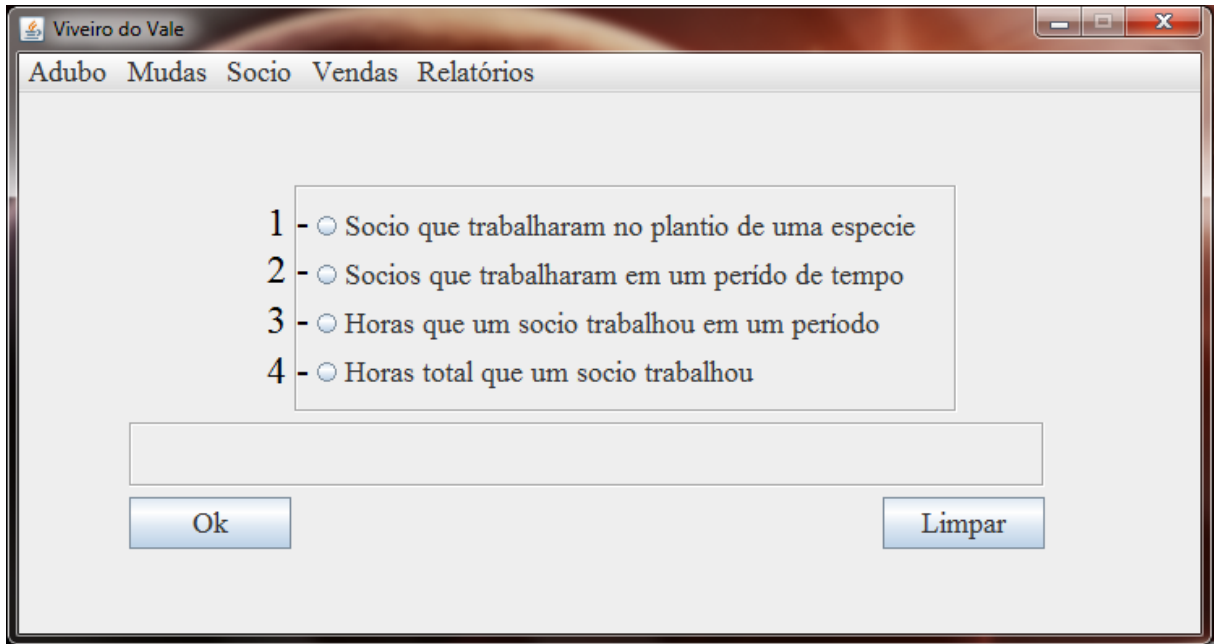
5.9.2. Relatório sobre mudas



Para estes relatórios com relação à mudas temos a classe `RelatorioMudas`, no qual cada um chama seu método:

1. Método chamado: [*`relatorioMudasEspecie`*](#).
 - Mostra as datas e a quantidade plantada de uma determinada espécie.
2. Método chamado: [*`relatorioMudasData`*](#).
 - Mostra as espécies e a quantidade plantada em um determinado período de tempo.
3. Método chamado: [*`relatorioMudasSocio`*](#).
 - Mostra as espécies, a data e a quantidade de mudas plantadas, no qual um determinado sócio participou do plantio.
4. Método chamado: [*`relatorioMudasAllUntilToday`*](#).
 - Mostra as datas e a quantidade da plantação de todas espécies.

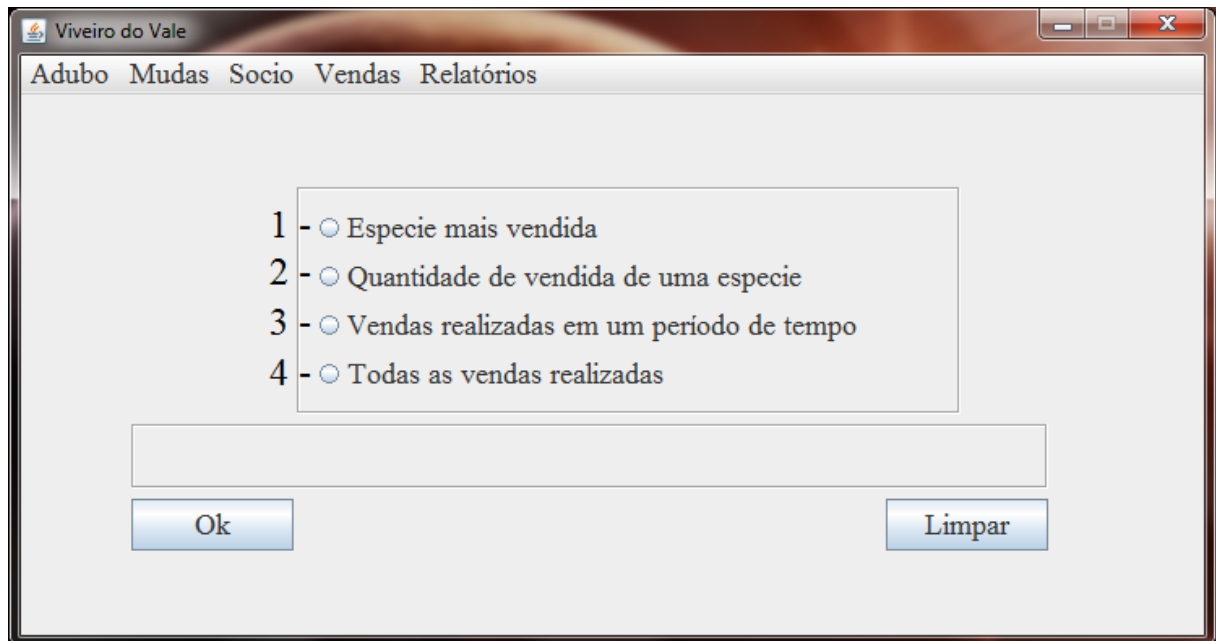
5.9.3. Relatório sobre sócios



Para estes relatórios com relação aos sócios temos a classe `RelatorioSocio`, no qual cada um chama seu método:

1. Método chamado: [`relatorioSocioEspecie`](#).
 - Mostra todas as espécies, data e quantidade de mudas em que um sócio participou do plantio.
2. Método chamado: [`relatorioSocioData`](#).
 - Mostra todas as espécies, data e quantidade de mudas que um sócio participou do plantio em determinado período de tempo.
3. Método chamado: [`relatorioSocioHorasData`](#).
 - Mostra a quantidade de horas, data e o nome da espécie das mudas em que um sócio participou do plantio.
4. Método chamado: [`relatorioSocioTotalHoras`](#).
 - Mostra o total de horas que um sócio trabalhou em todo período.

5.9.4. Relatório sobre vendas



Para estes relatórios com as vendas temos a classe `RelatorioVenda`, no qual cada um chama seu método:

1. Método chamado: [`relatorioVendaMais`](#).
 - Mostra a quantidade da espécie mais vendida.
2. Método chamado: [`relatorioVendaQuant`](#).
 - Mostra as datas das vendas em uma determinada espécie estava incluída.
3. Método chamado: [`relatorioVendaData`](#).
 - Mostra as espécies e as datas das vendas realizadas em um período de tempo.
4. Método chamado: [`relatorioVendaAll`](#).
 - Mostra todas as vendas realizadas.

A proposta para deste trabalho é desenvolver um sistema para um viveiro que vende mudas de plantas nativas do cerrado. Este viveiro trabalha em forma de sociedade, onde cada sócio tem uma porcentagem do que produziu no seu período de trabalho. Todos os sócios são responsáveis pelas vendas, não havendo comissão por vendas realizadas.

O sistema deve armazenar dados referentes às mudas disponíveis para venda, dados referentes aos adubos utilizados no plantio das mudas e dados referentes às horas trabalhadas de cada sócio. O sistema deve ser também capaz de gerar relatórios referentes aos adubos gastos, às mudas plantadas, às horas trabalhadas e às vendas. Sobre as mudas deseja-

se guardar o nome de cada espécie, a quantidade de mudas disponíveis para venda, o dia do plantio. Sobre os adubos deseja-se guardar o nome de cada adubo e a quantidade disponível para o plantio das mudas. O sistema deve informar ser capaz de gerar vários tipos de relatório para cada informação (adubo, mudas, sócio, vendas).

Interações com a base de dados

6.1. Classe Conexão

```
21 public static Connection getCon() {
22     try {
23         return DriverManager.getConnection("jdbc:mysql://localhost/VIVEIRO", "root", "");
24     } catch (SQLException ex) {
25         JOptionPane.showMessageDialog(null, "Erro na conexão!\n" + ex.getMessage());
26         JOptionPane.showMessageDialog(null, "O programa será fechado!");
27         System.exit(1);
28         return null;
29     }
30 }
```

Contem um único método estático responsável por criar a conexão com a base de dados.

6.2. Classe AduboDAO

6.2.1. Método *inserir*

```
38 public static int insere (Adubo novo) {
39     if ((con = Conexao.getCon()) != null) {
40         sql = "INSERT INTO ADUBO (nome, quantidade, preco) VALUES (?, ?, ?)" //
41             + "ON DUPLICATE KEY UPDATE nome = ?, quantidade = quantidade + ?, preco = ?";
42         try {
43             stm = con.prepareStatement(sql);
44
45             stm.setString(1, novo.getNome());
46             stm.setFloat(2, novo.getQuant());
47             stm.setFloat(3, novo.getPreco());
48
49             stm.setString(4, novo.getNome());
50             stm.setFloat(5, novo.getQuant());
51             stm.setFloat(6, novo.getPreco());
52
53             y = stm.executeUpdate()
54
55             stm.close();
56             con.close();
57
58             return y; //Retorna o resultado da execução
59         } catch (SQLException ex) {
60             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
61             return 0; //Indica erro
62         }
63     } else return 0; //Indica erro
64 }
```

Método responsável por inserir um novo adubo na base. Caso o adubo já exista, sua quantidade é atualizada somando-se a quantidade existente com a nova quantidade e o preço é substituído pelo novo preço.

6.2.2. Método *realizaBusca*

```
77 private static ArrayList<Adubo> realizaBusca(String comando) {
78     if ((con = Conexao.getCon()) != null) {
79         try {
80             stm = con.prepareStatement(comando);
81
82             rs = stm.executeQuery();
83
84             ArrayList<Adubo> novo = new ArrayList();
85
86             while (rs.next()) {
87
88                 Adubo aux = new Adubo();
89                 aux.setNome(rs.getString("nome"));
90                 aux.setQuant(rs.getFloat("quantidade"));
91                 aux.setPreco(rs.getFloat("preco"));
92                 novo.add(aux);
93             }
94
95             (novo.isEmpty()){
96                 novo = null;
97             }
98
99             stm.close();
100            con.close();
101
102            return novo;
103        } catch (SQLException ex) {
104            JOptionPane.showMessageDialog(null, "Erro na busca!\n" + ex.getMessage());
105            return null;
106        }
107    } else {
108        return null;
109    }
110 }
```

Método responsável por executar diferentes busca de adubos a partir de um comando, porem todos os resultados devem conter os mesmos atributos de retorno. Seu parâmetro de entrada é o comando a ser executado na base.

6.2.3. Método *busca* (sem parâmetro)

```
117 public static ArrayList<Adubo> busca() {
118     sql = "SELECT * FROM ADUBO";
119     return realizaBusca(sql);
120 }
```

Método responsável por buscar todos os adubos disponíveis. Seu retorno é a chamada ao método [realizaBusca](#), passando o comando a ser executado (String sql).

6.2.4. Método *busca* (com parâmetro)

```
128 public static ArrayList<Adubo> busca(String nome) {  
129     sql = "SELECT * FROM ADUBO WHERE nome like '%" + nome + "%'";  
130     return realizaBusca(sql);  
131 }
```

Método responsável por buscar um adubo específico. Seu parâmetro de entrada é o nome do adubo busca. Seu retorno é a chamada ao método [realizaBusca](#) passando o comando de busca (String sql).

6.2.5. Método *buscaPreco*

```
140 public static ArrayList<Adubo> buscaPreco(String nome) {  
141     sql = "SELECT nome_ad_fk as nome, sum(quant_ad) as quantidade, preco "  
142         + "FROM adubo_gasto, adubo "  
143         + "WHERE especie_fk = '" + nome + "' AND nome = nome_ad_fk "  
144         + "GROUP BY nome_ad_fk "  
145         + "ORDER BY data";  
146     return realizaBusca(sql);  
147 }
```

Método responsável por buscar todos os adubos gasto na produção de uma determinada espécie. Seu parâmetro de entrada é o nome da espécie. Seu retorno é a chamada ao método [realizaBusca](#) passando o comando de busca (String sql).

6.3. Classe AduboGastoDAO

6.3.1. Método *insere*

```
29 public static boolean insere(AduboGasto novo) {
30     x = true;
31     if ((con = Conexao.getCon()) != null) {
32         sql = "INSERT INTO ADUBO_GASTO VALUES (?, ?, ?, ?)";
33         try {
34             stm = con.prepareStatement(sql);
35
36             stm.setString(1, novo.getEspecie());
37             stm.setString(2, novo.getNome_ad());
38             stm.setFloat(3, novo.getQuant_ad());
39             stm.setString(4, novo.getData());
40             x = stm.execute();
41             stm.close();
42             con.close();
43             return x;
44         } catch (SQLException ex) {
45             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
46             return false;
47         }
48     } else {
49         return false;
50     }
51 }
```

Método responsável por inserir um novo registro de adubo gasto na base. Nesta inserção são gravados o nome da espécie do plantio, o nome do adubo utilizado, a quantidade de adubo utilizado e a data do plantio.

6.4. Classe MudasDAO

6.4.1. Método *insere*

```
36 public static int insere (Mudas novo) {
37     if ((con = Conexao.getCon()) != null) {
38         sql = "INSERT INTO mudas (nome) VALUES (?)";
39         try {
40             stm = con.prepareStatement(sql);
41
42             stm.setString(1, novo.getEspecie());
43
44             y = stm.executeUpdate();
45
46             stm.close();
47             con.close();
48
49             return y;
50         } catch (SQLException ex) {
51             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
52             return 0;
53         }
54     } else {
55         return 0;
56     }
57 }
```

Método responsável por inserir uma nova espécie na base.

6.4.2. Método *realizaBusca*

```
59 private static ArrayList<Mudas> realizaBusca(String busca) {
60     if ((con = Conexao.getCon()) != null) {
61         try {
62             stm = con.prepareStatement(busca);
63             rs = stm.executeQuery();
64             ArrayList<Mudas> novo = new ArrayList();
65             while (rs.next()) {
66                 novo.add(new Mudas(rs.getString("nome"), rs.getInt("quantidade")));
67             }
68             if (novo.isEmpty())
69                 novo = null;
70             stm.close();
71             con.close();
72             return novo;
73         } catch (SQLException ex) {
74             JOptionPane.showMessageDialog(null, "Erro na busca!\n" + ex.getMessage());
75             return null;
76         }
77     } else return null;
78 }
```

Método responsável por executar diferentes buscas de mudas a partir de um comando, porem todos os resultados devem ter os mesmos atributos de retorno. Seu parâmetro de entrada é o comando de busca a ser executado na base.

6.4.3. Método *busca* (sem parâmetro)

```
95 public static ArrayList<Mudas> busca() {
96     sql = "SELECT * FROM MUDAS";
97     return realizaBusca(sql);
98 }
```

Método responsável por buscar todas as mudas disponíveis. Seu retorno é a chamada ao método [realizaBusca](#), passando o comando a ser executado (String sql).

6.4.4. Método *busca* (com parâmetro)

```
105 public static ArrayList<Mudas> busca(String nome) {
106     sql = "SELECT * FROM MUDAS WHERE nome like '%" + nome + "%'";
107     return realizaBusca(sql);
108 }
```

Método responsável por buscar as mudas disponíveis de uma determinada espécie. Seu parâmetro de entrada é o nome da espécie. Seu retorno é a chamada ao método [realizaBusca](#) passando o comando de busca (String sql).

6.5. Classe MudasPlantadosDAO

6.5.1. Método *insere*

```
37 public static int insere(MudasPlantadas novo) {
38     if ((con = Conexao.getCon()) != null) {
39         sql = "INSERT INTO mudas_plantadas (especie, data, quant) VALUES (?, ?, ?)";
40         try {
41             stm = con.prepareStatement(sql);
42
43             stm.setString(1, novo.getEspecie());
44             stm.setString(2, novo.getData());
45             stm.setInt(3, novo.getQuant());
46             y = stm.executeUpdate();
47             stm.close();
48             con.close();
49
50             return y;
51         } catch (SQLException ex) {
52             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
53             return 0;
54         }
55     } else {
56         return 0;
57     }
58 }
```

Método responsável por inserir um novo registro de mudas plantadas na base. Nesta inserção são gravados o nome da espécie do plantio, a quantidade plantada e a data do plantio.

6.5.2. Método *buscaTotal*

```
62 public static ArrayList<MudasPlantadas> buscaTotal(String especie) {
63     if ((con = Conexao.getCon()) != null) {
64         sql = "SELECT especie, data, sum(quant) as quantidade "
65             + "FROM mudas_plantadas "
66             + "WHERE especie = " + especie + " "
67             + "ORDER BY data";
68         try {
69             stm = con.prepareStatement(sql);
70
71             rs = stm.executeQuery();
72
73             ArrayList<MudasPlantadas> novo = new ArrayList();
74             while (rs.next()) {
75                 String esp = rs.getString("especie");
76                 String dat = rs.getString("data");
77                 int qtd = rs.getInt("quantidade");
78                 MudasPlantadas aux = new MudasPlantadas(qtd, esp, dat);
79                 novo.add(aux);
80             }
81
82             if (novo.isEmpty()) {
83                 novo = null;
84             }
85
86             stm.close();
87             con.close();
88
89             return novo;
90         } catch (SQLException ex) {
91             JOptionPane.showMessageDialog(null, "Erro na busca!" + ex.getMessage());
92             return null;
93         }
94     } else {
95         return null;
96     }
97 }
```

Método responsável por busca o total de mudas plantadas de uma determinada espécie. Seu parâmetro de entrada é o nome da espécie.

6.6. Classe S cioDAO

6.6.1. M todo *insere*

```
30 public static int insere(Socio novo) {
31     if ((con = Conexao.getCon()) != null) {
32         sql = "INSERT INTO SOCIO VALUES (?, ?, ?, ?, ?, ?, ?, ?) "
33             + "ON DUPLICATE KEY UPDATE "
34             + "cpf = ?, nome = ?, rua = ?, numero = ?, bairro = ?, "
35             + "complemento = ?, cidade = ?, estado = ?, telefone = ?";
36         try {
37             stm = con.prepareStatement(sql);
38
39             stm.setString(1, novo.getCpf());
40             stm.setString(2, novo.getNome());
41             stm.setString(3, novo.getRua());
42             stm.setInt(4, novo.getNumero());
43             stm.setString(5, novo.getComplemento());
44             stm.setString(6, novo.getBairro());
45             stm.setString(7, novo.getCidade());
46             stm.setInt(8, novo.getEstado());
47             stm.setString(9, novo.getTelefone());
48
49             stm.setString(10, novo.getCpf());
50             stm.setString(11, novo.getNome());
51             stm.setString(12, novo.getRua());
52             stm.setInt(13, novo.getNumero());
53             stm.setString(14, novo.getComplemento());
54             stm.setString(15, novo.getBairro());
55             stm.setString(16, novo.getCidade());
56             stm.setInt(17, novo.getEstado());
57             stm.setString(18, novo.getTelefone());
58
59             y = stm.executeUpdate();
60
61             stm.close();
62             con.close();
63
64             return y;
65         } catch (SQLException ex) {
66             JOptionPane.showMessageDialog(null, "Erro na inser  o!\n" + ex.getMessage());
67             return 0;
68         }
69     } else {
70         return 0;
71     }
72 }
```

M todo respons vel por inserir um novo s cio na base. Neste caso o m todo retorna 1 (um). Caso o s cio j  exista, seus dados s o atualizados e o m todo retorna 2 (dois).

6.6.2. Método *realizaBusca*

```
74 private static ArrayList<Socio> realizaBusca(String busca) {
75     if ((con = Conexao.getCon()) != null) {
76         try {
77             stm = con.prepareStatement(busca);
78
79             rs = stm.executeQuery();
80
81             ArrayList<Socio> novo = new ArrayList();
82             while (rs.next()) {
83                 String cpf = rs.getString("cpf");
84                 String nom = rs.getString("nome");
85                 String rua = rs.getString("rua");
86                 int numero = rs.getInt("numero");
87                 String bai = rs.getString("bairro");
88                 String cmp = rs.getString("complemento");
89                 String cty = rs.getString("cidade");
90                 int estado = rs.getInt("estado");
91                 String tfl = rs.getString("telefone");
92                 Sócio aux = new Sócio(numero, estado, cpf, nom, rua, bai, cmp, cty, tfl);
93                 novo.add(aux);
94             }
95
96             if (novo.isEmpty()) {
97                 novo = null;
98             }
99
100            stm.close();
101            con.close();
102
103            return novo;
104        } catch (SQLException ex) {
105            JOptionPane.showMessageDialog(null, "Erro na busca!\n" + ex.getMessage());
106            return null;
107        }
108    } else {
109        return null;
110    }
111 }
```

Método responsável por executar diferentes buscas de sócios a partir de um comando, porem todos os resultados devem ter os mesmos atributos de retorno. Seu parâmetro de entrada é o comando de busca a ser executado na base.

6.6.3. Método *buscaAll*

```
113 public static ArrayList buscaAll() {
114     sql = "SELECT * FROM SOCIO ORDER BY nome";
115     return realizaBusca(sql);
116 }
```

Método responsável por buscar todos sócios cadastrados na base. Seu retorno é a chamada ao método [*realizaBusca*](#) passando o comando a ser executado.

6.6.4. Método *buscaPorCpf*

```
118 public static ArrayList buscaPorCpf(String cpf) {
119     sql = "SELECT * FROM SOCIO WHERE cpf = " + cpf + "";
120     return realizaBusca(sql);
121 }
```

Método responsável por buscar um sócio específico. Seu parâmetro de entrada é o CPF do sócio procurado. Seu retorno é a chamada ao método [*realizaBusca*](#) passando o comando a ser executado.

6.7. Classe SócioTrabalhouDAO

6.7.1. Método *insere*

```
29 public static boolean insere (SocioTrabalhou novo) {
30     if ((con = Conexao.getCon()) != null) {
31         sql = "INSERT INTO socio_trabalhou VALUES (?, ?, ?, ?)";
32         try {
33             stm = con.prepareStatement(sql);
34
35             stm.setString(1, novo.getCpf());
36             stm.setString(2, novo.getEspecie());
37             stm.setString(3, novo.getData());
38             stm.setString(4, novo.getHora());
39
40             x = stm.execute();
41
42             stm.close();
43             con.close();
44
45             return x;
46         } catch (SQLException ex) {
47             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
48             return false;
49         }
50     } else {
51         return false;
52     }
53 }
```

Método responsável por inserir um novo registro na base. Nesta inserção são gravados uma referencia para o sócio, a espécie plantada as horas trabalhadas e a data do plantio.

6.8. Classe VendaDAO

6.8.1. Método *insere*

```
35 public static boolean insere (Venda novo) {
36     if ((con = Conexao.getCon()) != null) {
37         sql = "INSERT INTO venda (numero, data) VALUES (?, ?)";
38         try {
39             stm = con.prepareStatement(sql);
40
41             stm.setInt(1, novo.getNumero());
42             stm.setString(2, novo.getData());
43
44             x = stm.execute();
45
46             stm.close();
47             con.close();
48
49             return x;
50         } catch (SQLException ex) {
51             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
52             return false;
53         }
54     } else {
55         return false;
56     }
57 }
```

Método responsável por inserir um novo registro de venda na base.

6.8.2. Método *proxNumero*

```
114 public static int proxNumero() {
115     if ((con = Conexao.getCon()) != null) {
116         int i = 0, j;
117         sql = "SELECT numero FROM venda";
118         try {
119             stm = con.prepareStatement(sql);
120             rs = stm.executeQuery();
121             while (rs.next()) {
122                 j = rs.getInt("numero");
123                 if (i + 1 != j)
124                     break;
125                 i++;
126             }
127             stm.close();
128             con.close();
129             return i + 1;
130         } catch (SQLException ex) {
131             JOptionPane.showMessageDialog(null, "Erro na busca do número!\n" + ex.getMessage());
132             return 0;
133         }
134     } else return 0;
135 }
```

Método responsável por buscar o próximo número de venda.

6.9. Classe VendaIncluiDAO

6.9.1. Método *inserir*

```
34 public static boolean inserir (VendaInclui novo) {
35     if ((con = Conexao.getCon()) != null) {
36         sql = "INSERT INTO venda_inclui VALUES (?, ?, ?, ?)";
37         try {
38             stm = con.prepareStatement(sql);
39
40             stm.setInt(1, novo.getNumero_venda());
41             stm.setString(2, novo.getEspecie());
42             stm.setInt(3, novo.getQuant());
43             stm.setFloat(4, novo.getPreco());
44
45             x = stm.execute();
46
47             stm.close();
48             con.close();
49
50             return x;
51         } catch (SQLException ex) {
52             JOptionPane.showMessageDialog(null, "Erro na inserção!\n" + ex.getMessage());
53             return false;
54         }
55     } else {
56         return false;
57     }
58 }
```

Método responsável por registra às mudas inclusas em uma venda.

6.10. Classe RelatórioAdubo

6.10.1. Método *relatorioAdubo*

```
25 private static ArrayList<Adubo> relatorioAdubo(String comando) {
26     if ((con = Conexao.getCon()) != null) {
27         try {
28             stm = con.prepareStatement(comando);
29             rs = stm.executeQuery();
30             ArrayList<Adubo> novo = new ArrayList();
31             Adubo aux;
32             while (rs.next()) {
33                 aux = new Adubo(rs.getString("nomeAdubo"), rs.getFloat("total"), rs.getFloat("preco"));
34                 novo.add(aux);
35             }
36             if (novo.isEmpty())
37                 novo = null;
38             stm.close();
39             con.close();
40             return novo;
41         } catch (SQLException ex) {
42             JOptionPane.showMessageDialog(null, "Erro no relatório de adubo!\n" + ex.getMessage());
43             return null;
44         }
45     } else return null;
46 }
```

Método responsável por executar todos os comandos de busca referentes aos relatórios sobre adubos.

6.10.2. Método *relatorioAduboEspecie*

```
53 public static ArrayList<Adubo> relatorioAduboEspecie(String especie) {
54     sql = "SELECT nome_ad_fk as nomeAdubo, sum(quant_ad) AS total, "
55         + "sum( ( (SELECT preco FROM adubo WHERE nome = nomeAdubo) * quant_ad) ) AS preco "
56         + "FROM adubo_gasto "
57         + "WHERE especie_fk = '" + especie + "' "
58         + "GROUP BY nome_ad_fk";
59     return relatorioAdubo(sql);
60 }
```

6.10.3. Método *relatorioAduboData*

```
62 public static ArrayList<Adubo> relatorioAduboData(String di, String df) {
63     sql = "SELECT nome_ad_fk as nomeAdubo, sum(quant_ad) AS total, "
64         + "sum(((SELECT preco FROM adubo WHERE nome = nomeAdubo) * quant_ad) )AS preco "
65         + "FROM adubo_gasto "
66         + "WHERE data >= '" + di + "' and data <= '" + df + "'"
67         + "GROUP BY nome_ad_fk";
68     return relatorioAdubo(sql);
69 }
```

6.10.4. Método *relatorioAduboGasto*

```
71 public static ArrayList<Adubo> relatorioAduboGasto() {
72     sql = "SELECT nome_ad_fk as nomeAdubo, sum(quant_ad) AS total, "
73         + "sum(((SELECT preco FROM adubo WHERE nome = nomeAdubo) * quant_ad) )AS preco "
74         + "FROM adubo_gasto "
75         + "WHERE especie_fk IN ("
76         + "    SELECT nome "
77         + "    FROM mudas "
78         + "    WHERE quantidade <> 0 "
79         + ") "
80         + "GROUP BY nome_ad_fk";
81     return relatorioAdubo(sql);
82 }
```

6.10.5. Método *relatorioAduboAllUntilToday*

```
84 public static ArrayList<Adubo> relatorioAduboAllUntilToday() {
85     sql = "SELECT nome_ad_fk as nomeAdubo, sum(quant_ad) AS total, "
86         + "sum(((SELECT preco FROM adubo WHERE nome = nomeAdubo) * quant_ad) )AS preco "
87         + "FROM adubo_gasto "
88         + "GROUP BY nome_ad_fk";
89     return relatorioAdubo(sql);
90 }
```

6.11. Classe RelatórioMudas

6.11.1. Método *relatorioMudas*

```
25 private static ArrayList<Mudas> relatorioMudas(String comando) {
26     if ((con = Conexao.getCon()) != null) {
27         try {
28             stm = con.prepareStatement(comando);
29
30             rs = stm.executeQuery();
31
32             ArrayList<Mudas> novo = new ArrayList();
33             while (rs.next()) {
34                 Mudas aux = new Mudas(rs.getString("data"), rs.getString("especie_fk"), rs.getInt("quant"));
35                 novo.add(aux);
36             }
37             if (novo.isEmpty()) {
38                 novo = null;
39             }
40
41             stm.close();
42             con.close();
43             return novo;
44         } catch (SQLException ex) {
45             JOptionPane.showMessageDialog(null, "Erro no relatório de adubo!!\n" + ex.getMessage());
46             return null;
47         }
48     } else {
49         return null;
50     }
```

Método responsável por executar todos os comandos de busca referentes aos relatórios sobre mudas.

6.11.2. Método *relatorioMudasEspecie*

```
53 public static ArrayList<Mudas> relatorioMudasEspecie(String especie) {
54     sql = "SELECT ag.especie_fk, ag.data, quant "
55         + "FROM adubo_gasto AS ag, mudas_plantadas AS mp "
56         + "WHERE ag.especie_fk = '" + especie + "'"
57         + "GROUP BY ag.data "
58         + "ORDER BY ag.data";
59     return relatorioMudas(sql);
60 }
```

6.11.3. Método *relatorioMudasData*

```
62 public static ArrayList<Mudas> relatorioMudasData(String di, String df) {
63     sql = "SELECT ag.especie_fk, ag.data, quant "
64         + "FROM adubo_gasto AS ag, mudas_plantadas AS mp "
65         + "WHERE ag.data >= '" + di + "'" AND ag.data <= '" + df + "'"
66         + "GROUP BY ag.data, ag.especie_fk "
67         + "ORDER BY ag.data";
68     return relatorioMudas(sql);
69 }
```

6.11.4. Método *relatorioMudasSocio*

```
71 public static ArrayList<Mudas> relatorioMudasSocio(String cpf) {
72     sql = "SELECT especie_fk, st.data, quant "
73         + "FROM socio_trabalhou as st, mudas_plantadas as mp "
74         + "WHERE cpf_fk = '" + cpf + "' AND st.data = mp.data AND st.especie_fk = mp.especie "
75         + "ORDER BY data, especie_fk";
76     return relatorioMudas(sql);
77 }
```

6.11.5. Método *relatorioMudasAllUntilToday*

```
79 public static ArrayList<Mudas> relatorioMudasAllUntilToday() {
80     sql = "SELECT ag.especie_fk, ag.data, quant "
81         + "FROM adubo_gasto AS ag, mudas_plantadas AS mp "
82         + "GROUP BY data, especie_fk "
83         + "ORDER BY data";
84     return relatorioMudas(sql);
85 }
```

6.12. Classe RelatórioSócio

6.12.1. Método *relatorioSocioEspecie*

```
25 public static ArrayList<SocioTrabalhou> relatorioSocioEspecie(String especie) {
26     if ((con = Conexao.getCon()) != null) {
27         sql = "SELECT nome, data, horas "
28             + "FROM socio_trabalhou, socio "
29             + "WHERE especie_fk = '" + especie + "' AND cpf = cpf_fk "
30             + "ORDER BY data, nome;";
31         try {
32             stm = con.prepareStatement(sql);
33             rs = stm.executeQuery();
34             ArrayList<SocioTrabalhou> novo = new ArrayList();
35             String data;
36             SocioTrabalhou aux;
37             while (rs.next()) {
38                 data = rs.getString("data").substring(8, 10) + "/"
39                     + rs.getString("data").substring(5, 7) + "/"
40                     + rs.getString("data").substring(0, 4);
41                 aux = new SocioTrabalhou();
42                 aux.setCpf(rs.getString("nome"));
43                 aux.setData(data);
44                 aux.setHora(rs.getString("horas").substring(0, 5));
45                 novo.add(aux);
46             }
47             if (novo.isEmpty())
48                 novo = null;
49             stm.close();
50             con.close();
51             return novo;
52         } catch (SQLException ex) {
53             JOptionPane.showMessageDialog(null, "Erro no relatório sócio 1.\n" + ex.getMessage());
54             return null;
55         }
56     }
57     else return null;
58 }
```

6.12.2. Método *relatorioSocioData*

```
63 public static ArrayList<SocioTrabalhou> relatorioSocioData(String di, String df) {
64     if ((con = Conexao.getCon()) != null) {
65         sql = "SELECT nome, especie_fk, horas, data "
66             + "FROM socio, socio_trabalhou "
67             + "WHERE data >= " + di + " AND data <= " + df + " AND cpf = cpf_fk "
68             + "ORDER BY data, especie_fk, nome;";
69         try {
70             stm = con.prepareStatement(sql);
71
72             rs = stm.executeQuery();
73
74             ArrayList<SocioTrabalhou> novo = new ArrayList();
75             String data;
76             while (rs.next()) {
77                 SocioTrabalhou aux;
78                 data = rs.getString("data").substring(8, 10) + "/"
79                     + rs.getString("data").substring(5, 7) + "/"
80                     + rs.getString("data").substring(0, 4);
81                 aux = new SocioTrabalhou();
82                 aux.setCpf(rs.getString("nome"));
83                 aux.setEspecie(rs.getString("especie_fk"));
84                 aux.setData(data);
85                 aux.setHora(rs.getString("horas"));
86                 novo.add(aux);
87             }
88
89             if (novo.isEmpty()) {
90                 novo = null;
91             }
92
93             stm.close();
94             con.close();
95
96             return novo;
97         } catch (SQLException ex) {
98             JOptionPane.showMessageDialog(null, "Erro no relatório sócio 2.\n" + ex.getMessage());
99             return null;
100         }
101     } else {
102         return null;
103     }
104 }
```

6.12.3. Método *relatorioSocioHorasData*

```
106 public static ArrayList<SocioTrabalhou> relatorioSocioHorasData(String cpf, String di, String df) {
107     if ((con = Conexao.getCon()) != null) {
108         sql = "SELECT especie_fk, horas, data "
109             + "FROM socio, socio_trabalhou "
110             + "WHERE cpf_fk = " + cpf + " AND data >= " + di + " AND "
111             + "data <= " + df + " AND cpf = cpf_fk "
112             + "ORDER BY data, especie_fk;";
113         try {
114             stm = con.prepareStatement(sql);
115
116             rs = stm.executeQuery();
117
118             ArrayList<SocioTrabalhou> novo = new ArrayList();
119             String data;
120             while (rs.next()) {
121                 SocioTrabalhou aux;
122                 data = rs.getString("data").substring(8, 10) + "/"
123                     + rs.getString("data").substring(5, 7) + "/"
124                     + rs.getString("data").substring(0, 4);
125                 aux = new SocioTrabalhou();
126                 aux.setEspecie(rs.getString("especie_fk"));
127                 aux.setData(data);
128                 aux.setHora(rs.getString("horas"));
129                 novo.add(aux);
130             }
131
132             if (novo.isEmpty()) {
133                 novo = null;
134             }
135
136             stm.close();
137             con.close();
138
139             return novo;
140         } catch (SQLException ex) {
141             JOptionPane.showMessageDialog(null, "Erro no relatório sócio.\n" + ex.getMessage());
142             return null;
143         }
144     } else {
145         return null;
146     }
147 }
```

6.12.4. Método *relatorioSocioTotalHoras*

```
149 public static String relatorioSocioTotalHoras(String cpf) {
150     int horas = 0, min = 0;
151     String total;
152     if ((con = Conexao.getCon()) != null) {
153         sql = "SELECT (sum(hour(horas))) as horas, sum(minute(horas)) as minutos "
154             + "FROM socio_trabalhou "
155             + "WHERE cpf_fk = " + cpf + ";";
156         try {
157             stm = con.prepareStatement(sql);
158
159             rs = stm.executeQuery();
160
161             while (rs.next()) {
162                 horas = rs.getInt("horas");
163                 min = rs.getInt("minutos");
164             }
165
166             stm.close();
167             con.close();
168
169             if ((min % 60) > 9) {
170                 total = String.valueOf(horas + ((int) min / 60)) + ":" + String.valueOf(min % 60);
171             } else {
172                 total = String.valueOf(horas + ((int) min / 60)) + ":0" + String.valueOf(min % 60);
173             }
174             return total;
175         } catch (SQLException ex) {
176             JOptionPane.showMessageDialog(null, "Erro no relatório sócio.\n" + ex.getMessage());
177             return null;
178         }
179     } else {
180         return null;
181     }
182 }
```

6.13. Classe RelatórioVenda

6.13.1. Método *relatorioVenda*

```
26 public static ArrayList<VendaInclui> relatorioVenda(String comando) {
27     if ((con = Conexao.getCon()) != null) {
28         try {
29             stm = con.prepareStatement(comando);
30
31             rs = stm.executeQuery();
32
33             ArrayList<VendaInclui> novo = new ArrayList();
34             while (rs.next()) {
35                 VendaInclui aux = new VendaInclui();
36                 aux.setNumero_venda(rs.getInt("num_venda"));
37                 aux.setEspecie(rs.getString("especie"));
38                 aux.setQuant(rs.getInt("quantidade"));
39                 aux.setPreco(rs.getFloat("preco"));
40                 novo.add(aux);
41             }
42             if (novo.isEmpty()) {
43                 novo = null;
44             }
45
46             stm.close();
47             con.close();
48
49             return novo;
50         } catch (SQLException ex) {
51             JOptionPane.showMessageDialog(null, "Erro na busca!" + ex.getMessage());
52             return null;
53         }
54     } else {
55         return null;
56     }
57 }
```

6.13.2. Método *relatorioVendaMais*

```
59 public static VendaInclui relatorioVendaMais() {
60     sql = "SELECT num_venda, especie, sum(quantidade) as quantidade, sum(preco) as preco "
61         + "FROM venda_inclui "
62         + "GROUP BY especie "
63         + "ORDER BY quantidade desc";
64     ArrayList<VendaInclui> aux = relatorioVenda(sql);
65     if (aux == null) {
66         return null;
67     } else {
68         return aux.get(0);
69     }
70 }
```


6.13.3. Método *relatorioVendaQuant*

```
72 public static VendaInclui relatorioVendaQuant(String especie) {
73     sql = "SELECT num_venda, especie, sum(quantidade) as quantidade, sum(preco) as preco "
74         + "FROM venda_inclui "
75         + "WHERE especie = '" + especie + "' "
76         + "GROUP BY especie";
77     ArrayList<VendaInclui> aux = relatorioVenda(sql);
78     if (aux == null) {
79         return null;
80     } else {
81         return aux.get(0);
82     }
83 }
```

6.13.4. Método *relatorioVendaData*

```
85 public static ArrayList<VendaInclui> relatorioVendaData(String di, String df) {
86     sql = "SELECT num_venda, especie, quantidade, preco "
87         + "FROM venda_inclui, venda "
88         + "WHERE data >= '" + di + "' AND data <= '" + df + "' AND numero = num_venda "
89         + "ORDER BY num_venda, especie";
90     return relatorioVenda(sql);
91 }
```

6.13.5. Método *relatorioVendaAll*

```
93 public static ArrayList<VendaInclui> relatorioVendaAll() {
94     sql = "SELECT num_venda, especie, quantidade, preco "
95         + "FROM venda_inclui, venda "
96         + "WHERE numero = num_venda "
97         + "ORDER BY num_venda, especie";
98     return relatorioVenda(sql);
99 }
```