



UNIVERSIDADE FEDERAL DE MATO GROSSO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMAS DE INFORMAÇÃO

RELATÓRIO DE ESTÁGIO SUPERVISIONADO
OTIMIZAÇÃO DE PERFORMANCE NO SISTEMA
GEAGROBIT

THAIS FERNANDA BUENO DA SILVA

CUIABÁ – MT

2015

UNIVERSIDADE FEDERAL DE MATO GROSSO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMAS DE INFORMAÇÃO

**RELATÓRIO DE ESTÁGIO SUPERVISIONADO
OTIMIZAÇÃO DE PERFORMANCE NO SISTEMA
GEAGROBIT**

THAIS FERNANDA BUENO DA SILVA

Relatório apresentado ao Instituto de
Computação da Universidade Federal de
Mato Grosso para obtenção do título de
Bacharel em Sistemas de Informação.

CUIABÁ – MT

2015

UNIVERSIDADE FEDERAL DE MATO GROSSO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
SISTEMAS DE INFORMAÇÃO

THAIS FERNANDA BUENO DA SILVA

Relatório de Estágio Supervisionado apresentado à Coordenação do Curso de Sistemas de Informação como uma das exigências para obtenção do título de Bacharel em Sistemas de Informação da Universidade Federal de Mato Grosso.

Aprovado por:

Prof. Nilton Hideki Takagi
Instituto de Computação
(COORDENADOR DE ESTÁGIOS)

Prof. Dr. Josiel Maimone de Figueiredo
Instituto de Computação
(ORIENTADOR - UFMT)

MSc. Marcus Vinicius da Silva Wagner
(SUPERVISOR - INOVABIT)

DEDICATÓRIA

À minha família, em especial aos meus pais e irmão pelo apoio e confiança.

AGRADECIMENTOS

Agradeço a minha família, em especial aos meus pais pela minha formação de caráter e pelos seus esforços a fim de me proporcionar educação.

Agradeço ao meu irmão pelo companheirismo e cumplicidade em todos os momentos.

Agradeço a todos os meus professores do Instituto de Computação por serem pacientes e dedicados quanto à transmissão de seus conhecimentos para nós alunos. Agradeço ao professor Nielsen pelo apoio durante toda a graduação, principalmente durante o intercâmbio realizado.

Agradeço a InovaBit e aos colegas de serviço pela oportunidade, ajuda e apoio durante a realização do estágio supervisionado.

Agradeço aos colegas de graduação, em especial aos amigos Julian Rodrigues e Weinne Santos por estarem sempre me apoiando e incentivando desde o início do curso.

Finalmente, agradeço aos amigos que conheci durante toda a minha vida e ao meu namorado por sempre me incentivar em meus projetos e me amparar em momentos difíceis.

A vocês, muito obrigada!

SUMÁRIO

LISTA DE FIGURAS	8
LISTA DE QUADROS	9
LISTA DE ANEXOS.....	10
LISTA DE SIGLAS E ABREVIATURAS	11
RESUMO	12
INTRODUÇÃO	13
1. REVISÃO DE LITERATURA	15
1.1. ENGENHARIA DE <i>SOFTWARE</i>	15
1.1.1. REQUISITOS DE <i>SOFTWARE</i>	17
1.1.2. ESPECIFICAÇÃO DE REQUISITOS	18
1.1.3. PROJETO DE <i>SOFTWARE</i>	19
1.1.4. MODELAGEM CONCEITUAL	19
1.1.5. PROJETO ARQUITETURAL	19
1.2. BANCO DE DADOS.....	20
1.2.1. TIPOS DE BANCO DE DADOS	20
1.2.2. LINGUAGEM DE CONSULTA ESTRUTURADA	21
1.2.3. SISTEMAS DE GERENCIAMENTO DE BANCO DE DADOS	21
1.2.4. SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS MYSQL.....	22
1.2.4.1. OTIMIZAÇÃO EM BANCO DE DADOS MYSQL	25
1.2.4.2. OTIMIZAÇÃO DE CONSULTAS SQL	25
2. MATERIAIS, TÉCNICAS E MÉTODOS.....	32
2.1. MATERIAIS	32
2.2. TÉCNICAS E MÉTODOS.....	32
3. ESTUDO DE CASO	34
3.1. <i>FRAMEWORK</i> INOVABIT	34
3.1.1. CLASSE MONITOR.....	36
3.1.2. FILTRO DE INFORMAÇÕES	36
3.1.3. CLASSE FILTRAR.....	38
3.2. O SISTEMA GEAGROBIT	39
3.2.1. DEFINIÇÃO DO PROBLEMA DE PERFORMANCE.....	40
3.3. MÓDULOS "PAGAMENTO DE FRETE" E "CONTAS A PAGAR E RECEBER ...	41
3.3.1. MÓDULO "CONTAS A PAGAR E RECEBER"	41
3.3.2. MÓDULO "PAGAMENTO DE FRETE"	48
4. DIFICULDADES ENCONTRADAS	56
5. CONSIDERAÇÕES FINAIS	57

REFERÊNCIAS BIBLIOGRÁFICAS	58
ANEXOS	60

Lista de Figuras

FIGURA 1: EXEMPLO DE DECLARAÇÃO SQL QUE NÃO UTILIZA ÍNDICE. NA PRIMEIRA DECLARAÇÃO USA-SE %, NA SEGUNDA A <i>STRING</i> DE COMPARAÇÃO NÃO É CONSTANTE (MYSQL).....	27
FIGURA 2: ARQUIVO DE <i>LOG</i> DE CONSULTAS LENTAS (DINIZ, 2011).....	28
FIGURA 3: EXEMPLO DE UTILIZAÇÃO DA CLÁUSULA <i>EXPLAIN</i> NO PHPMYADMIN.	28
FIGURA 4: ARQUITETURA DOS SISTEMAS QUE UTILIZAM O <i>FRAMEWORK</i> INOVABIT.	35
FIGURA 5: CLASSE MONITOR DO <i>FRAMEWORK</i> INOVABIT.	36
FIGURA 6: TABELAS NO BANCO DE DADOS MENCIONADAS NO MÓDULO MONITOR.	36
FIGURA 7: TRECHO DO FILTRO DO MÓDULO "CONTAS A PAGAR E RECEBER" SISTEMA GEAGROBIT.....	37
FIGURA 8: CÓDIGO DA FUNÇÃO "FILTRARSQL" DO MÓDULO BANCOS DO SISTEMA GEAGROBIT, QUE UTILIZA A CLASSE FILTRAR DO <i>FRAMEWORK</i>	38
FIGURA 9: RETORNO DA FUNÇÃO "FILTRARSQL" DO MÓDULO BANCO DO SISTEMA GEAGROBIT.	39
FIGURA 10: TELA INICIAL DO SISTEMA GEAGROBIT APÓS REALIZAÇÃO DE <i>LOGIN</i>	39
FIGURA 11: FLUXO DE PROCESSOS DO SISTEMA GEAGROBIT.	40
FIGURA 12: FLUXO DO MÓDULO "CONTAS A PAGAR E RECEBER" DO SISTEMA GEAGROBIT.....	42
FIGURA 13: DIAGRAMA DE CASO DE USO DO MÓDULO "CONTAS A PAGAR E RECEBER".	43
FIGURA 14: DIAGRAMA DE CONCEITOS DO MÓDULO "CONTAS A PAGAR E RECEBER".....	44
FIGURA 15: USO DA CLASSE MONITOR NO FILTRO DO MÓDULO "CONTAS A PAGAR E RECEBER" DO SISTEMA GEAGROBIT.....	45
FIGURA 16: FUNÇÃO QUE GERA CÓDIGO SQL APÓS MODIFICAÇÃO DE ATRIBUTO PARA VERIFICAÇÃO DE USO NA FUNÇÃO <i>GETLEGENDA</i> A FIM DE DIMINUIR A QUANTIDADE DE DADOS RETORNADOS, CONFORME NECESSIDADE DE USO.....	46
FIGURA 17: ALTERAÇÃO NA FUNÇÃO " <i>GETDADOORIGEMMESCLAGEMSQL</i> " PARA LIMITAR A QUANTIDADE DE REGISTROS RETORNADAS, OTIMIZANDO A <i>QUERY</i> PRINCIPAL DA FUNÇÃO "FILTRARSQL".....	47
FIGURA 18: GRÁFICO ILUSTRANDO A MELHORIA NA PERFORMANCE DO MÓDULO "CONTAS A PAGAR E RECEBER APÓS AJUSTES NA <i>SUBQUERY</i>	48
FIGURA 19: DIAGRAMA DE CASO DE USO DO MÓDULO "PAGAMENTO DE FRETE".....	49
FIGURA 20: DIAGRAMA DE CONCEITOS DO MÓDULO "CONTAS A PAGAR E RECEBER".....	50
FIGURA 21: USO DA CLASSE MONITOR NO FILTRO DO MÓDULO "PAGAMENTO DE FRETE" DO SISTEMA GEAGROBIT.	51
FIGURA 22: EXEMPLO DA ESTRUTURA DA CONSULTA SQL DA FUNÇÃO FILTRAR EM "PAGAMENTO DE FRETE".	52
FIGURA 23: UTILIZAÇÃO DA FUNÇÃO <i>GETSTRINGSQL</i> DO <i>FRAMEWORK</i> INOVABIT PARA ADIÇÃO DE CONDIÇÕES NAS CLÁUSULAS <i>WHERE</i>	53
FIGURA 24: COMPARAÇÃO DO DESEMPENHO DAS CONSULTAS SQL DA FUNÇÃO FILTRAR DO MÓDULO "PAGAMENTO DE FRETE" DO SISTEMA GEAGROBIT EM AMBIENTE DE TESTE.	53
FIGURA 25: COMPARAÇÃO DO TEMPO DE EXECUÇÃO DA CONSULTA DA FUNÇÃO FILTRAR NO MÓDULO "PAGAMENTO DE FRETE".	55

Lista de Quadros

QUADRO 1: SIGNIFICADO DOS BOTÕES NOS CAMPOS DO FILTRO NO SISTEMA GEAGROBIT	37
QUADRO 2: RESULTADO DO MONITORAMENTO DO FILTRO EM "PAGAMENTO DE FRETE" DO SISTEMA GEAGROBIT NO SERVIDOR LOCAL PARA VERIFICAR O PROBLEMA INFORMADO.....	51
QUADRO 3: ÍNDICES DA TABELA FRETE_PAGAMENTO_REGISTRO	54

Lista de Anexos

ANEXO 1: CLASSE MONITOR	60
ANEXO 2: LISTA DE TABELAS RELACIONADAS AO MÓDULO "CONTAS A PAGAR E RECEBER"	61
ANEXO 3: LISTA DE TABELAS RELACIONADAS AO MÓDULO "PAGAMENTO DE FRETE"	62
ANEXO 4: CASOS DE USO DO MÓDULO "CONTAS A PAGAR E RECEBER"	63
ANEXO 5: CASOS DE USO DO MÓDULO "PAGAMENTO DE FRETE"	70

Lista de Siglas e Abreviaturas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade, do inglês: <i>Atomicity, Consistency, Isolation, Durability</i>
AJAX	Javascript Assíncrono e XML, do inglês <i>Asynchronous Javascript and XML</i>
BLOB	Do inglês <i>Binary Large Object</i> . É uma coleção de dados para o armazenamento de qualquer tipo de informações em formato binário, geralmente são objetos de imagem, áudio ou outros objetos multimídia, armazenada em uma tabela de um banco de dados.
FEBRABAN	Federação Brasileira de Bancos.
CNAB	Centro Nacional de Automação Bancária, núcleo da FEBRABAN responsável por padronizar as normas e layouts das cobranças eletrônicas.
CSV	Do inglês <i>comma-separated values</i> é um formato de arquivo que armazena dados tabulares em forma de texto simples separados por um caractere definido (mais comum utilizar vírgula como separador).
HD	Disco rígido, do inglês <i>Hard Disk</i> .
HTML	Linguagem de Marcação de Hipertexto, do inglês <i>HyperText Markup Language</i> .
ISAM	Método de Acesso Sequencial Indexado, do inglês <i>Indexed Sequential Access Method</i> .
JVM	Máquina Virtual Java, do inglês <i>Java Virtual Machine</i>
PHP	Sigla recursiva para "PHP: <i>Hypertext Preprocessor</i> ", uma linguagem de programação especialmente adequada para desenvolvimento <i>web</i> .
RAM	Memória de Acesso Aleatório, do inglês <i>Random Access Memory</i> .
SGBD	Sistema de Gerenciamento de Banco de Dados.
SQL	Linguagem de Consulta Estruturada, do inglês <i>Structured Query Language</i> .
URL	Do inglês <i>Uniform Resource Locator</i> , é o endereço de um recurso (como um arquivo, por exemplo), disponível em uma rede.
XML	Do inglês <i>eXtensible Markup Language</i> .

Resumo

Este documento apresenta as atividades desenvolvidas no período de estágio supervisionado da discente Thais Fernanda Bueno da Silva, acadêmica de Sistemas de Informação da Universidade Federal de Mato Grosso, sob supervisão de Marcus Vinicius da Silva Wagner e orientação do Prof. Josiel Maimone de Figueiredo. As atividades realizadas durante o estágio constituem-se no desenvolvimento de uma solução para a melhoria do desempenho de dois módulos do sistema de gestão de agronegócios GeAgroBit: "Pagamento de Frete" e "Contas a Pagar e Receber".

Introdução

A gestão de um sistema de produção agrícola apresenta diversos desafios devido a sua complexidade. Uma fazenda precisa gerir desde a sua produção, envolvendo etapas como planejamento de safra, receituários agrônômicos, embalagens controladas; até a gestão financeira, recursos humanos, contabilidade, emissão de nota fiscal e gestão da saída de produção.

A InovaBit, com o objetivo de suprir a demanda de um de seus clientes iniciou em 2010 o desenvolvimento de um sistema de gestão agrícola voltado para uma fazenda de grande porte. Inicialmente com poucos módulos, o sistema vem agregando novas funcionalidades continuamente desde então.

Atualmente o sistema não possui uma documentação consistente, o que leva novos e antigos desenvolvedores da empresa a encontrarem dificuldades para compreender o funcionamento do sistema durante o desenvolvimento de novas funcionalidades ou manutenção de código. Além disso, existem alguns problemas de projeto, como a falta de uma solução para o aumento da quantidade de dados que o sistema necessita manipular, o que gera a perda de desempenho em alguns módulos.

Desta forma, o objetivo deste trabalho é documentar de maneira sucinta dois módulos do sistema GeAgroBit como forma de apoio na compreensão do sistema e desenvolver uma solução para o problema referente a performance na manipulação de dados.

Pretende-se ainda que, através da elaboração deste trabalho o aluno em estágio aprenda os conceitos relacionados à engenharia de *software* e banco de dados, bem como a trabalhar com as ferramentas: NetBeans, Linguagem de programação PHP, banco de dados MySQL e a ferramenta PhpMyAdmin.

A partir desta introdução, este trabalho está organizado da seguinte forma: no Capítulo 1 é apresentada a revisão de literatura com as definições de engenharia de *software* e banco de dados utilizadas para o desenvolvimento da solução proposta; no Capítulo 2 são apresentados os materiais, técnicas e métodos utilizados na realização deste trabalho, no Capítulo 3 são apresentados os resultados atingidos, no Capítulo 4 são apresentadas as dificuldades encontradas no decorrer deste trabalho, no Capítulo 5

são apresentadas as conclusões obtidas e por fim as referências bibliográficas utilizadas como apoio a este trabalho e anexos.

1. REVISÃO DE LITERATURA

Para realizar o desenvolvimento deste trabalho fez-se necessário o levantamento de bibliografia de engenharia de *software* e banco de dados, sendo destacados a seguir os tópicos relevantes no resultado final do estágio supervisionado.

1.1. Engenharia de *Software*

A engenharia de *software* é a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de *software*. BOURQUE e FAIRLEY dividem a Engenharia de *Software* em quinze áreas de conhecimento listadas a seguir:

- Requisitos de *Software* (*Software Requirements*): Levantamento, análise, especificação e validação de requisitos de *software*, e gerenciamento de requisitos durante todo o ciclo de vida do produto de *software*.
- Projeto de *Software* (*Software Design*): Descrição da arquitetura de *software* isto é, como o *software* é decomposto e organizado em componentes, e as interfaces entre esses componentes.
- Construção de *Software* (*Software Construction*): Desenvolvimento do *software* a partir do projeto, resultando em um produto a ser testado.
- Teste de *Software* (*Software Testing*): É a verificação se o *software* desenvolvido fornece os comportamentos esperados usando casos de testes devidamente documentados.
- Manutenção de *Software* (*Software Maintenance*): A manutenção de *software* é um conjunto de atividades necessárias para fornecer suporte de baixo custo ao sistema. As atividades são executadas pouco antes da entrega e/ou após entrega.
- Gestão de Configuração de *Software* (*Software Configuration Management*): A Gestão de Configuração identifica a configuração de um sistema em diversos pontos no tempo a fim de controlar as alterações na

manutenção e configuração e assim verificar a conformidade com os requisitos ao longo do ciclo de vida de um sistema.

- Gestão de Engenharia de *Software* (*Software Engineering Management*): Realiza a aplicação de gestão de atividades de planejamento, coordenação, medição, monitoramento e emissão de relatórios com o objetivo de garantir que os produtos de *software* serão entregues de forma eficiente e eficaz.
- Engenharia de Processo de *Software* (*Software Engineering Process*): Conjunto de atividades relacionadas que transformam uma ou mais entradas em saídas, consumindo recursos para realizar a transformação. Os processos de engenharia de *software* estão preocupados com as atividades de trabalho realizadas para desenvolver, manter e operar o *software*, como requisitos, design, construção, testes, gerenciamento de configuração, e outros processos.
- Modelos e Métodos de Engenharia de *Software* (*Software Engineering Models and Methods*): Impõe uma estrutura de Engenharia de *Software* com o objetivo de fazer das atividades mais orientadas ao sucesso.
- Qualidade de *Software* (*Software Quality*): Esta área aborda definições e fornece uma visão geral das práticas, ferramentas e técnicas para a definição de qualidade de *software* e para avaliar o estado de qualidade de *software* durante o desenvolvimento, manutenção e implantação. Qualidade de *software* pode referir-se as características desejáveis de produtos de *software*, na medida em que um produto possui essas características, e aos processos, ferramentas e técnicas usadas para alcançar essas características, ou pode ser definida como a "capacidade de produto de *software* para satisfazer necessidades explícitas e implícitas nas condições especificadas" e como "o grau em que um produto de *software* atende aos requisitos estabelecidos".
- Prática Profissional de Engenharia de *Software* (*Software Engineering Professional Practice*): Esta área preocupa-se com o conhecimento, habilidades e atitudes que os engenheiros de *software* devem possuir para

a prática de engenharia de *software* de uma forma profissional, responsável e ética.

- Economia em Engenharia de *Software* (*Software Engineering Economics*): Esta área relaciona-se a tomada de decisões relacionadas com a engenharia de *software* em um contexto de negócios, preocupando-se com o alinhamento das decisões técnicas da Engenharia de *Software* com os objetivos de negócio da organização.
- Fundamentos da Computação (*Computing Foundations*): O conhecimento sobre o computador e seus princípios de *hardware* e *software* são importantes pois abrange o desenvolvimento e ambiente operacional em que o *software* é desenvolvido e executado.
- Fundamentos da Matemática (*Mathematical Foundations*): Auxilia os engenheiros de *software* a compreender a lógica usada em código nas linguagens de programação.
- Fundamentos da Engenharia (*Engineering Foundations*): Esta área está relacionada aos fundamentos de engenharia que incluem métodos empíricos e técnicas experimentais; análise estatística; medição; projeto de engenharia; modelagem, prototipagem e simulação e normas. Estes conhecimentos permitirão que os engenheiros de *software* desenvolvam e mantenham o *software* mais eficiente e eficaz.

A seguir serão explorados alguns pontos onde verificou-se a necessidade da busca de referencial teórico para desenvolvimento do trabalho.

1.1.1. Requisitos de *Software*

Os requisitos de um sistema são "descrições dos serviços fornecidos pelo sistema e suas restrições operacionais" (SOMMERVILLE, 2007). A área de engenharia de requisitos está relacionada ao levantamento, análise, especificação, validação e o gerenciamento de requisitos durante todo o ciclo de vida do desenvolvimento de um *software*. Os requisitos podem ser sobre o produto (uma restrição ou necessidade sobre o *software* a ser desenvolvido, como por exemplo, "O *software* deve verificar se o aluno possui todas as matérias de pré-requisitos cursadas antes de realizar a rematrícula"), ou sobre processos (restrição sobre o

desenvolvimento do *software*, como por exemplo, "O sistema deve ser desenvolvido usando a linguagem de programação JAVA") (BOURQUE e FAIRLEY, 2014).

Os requisitos são classificados como funcionais quando descrevem as funcionalidades que o *software* deve possuir. BOURQUE e FAIRLEY definem requisitos funcionais como aqueles em que "um conjunto finito de passos de teste pode ser escrito para validar o seu comportamento".

1.1.2. Especificação de Requisitos

Em engenharia de *software*, "especificação de requisitos de *software*" se refere à produção de um documento que pode ser sistematicamente revisto, avaliado e aprovado. Para sistemas complexos geralmente se produz três diferentes tipos de documentos: definição do sistema, requisitos de sistema e requisitos de *software*. Para produtos de *software* simples geralmente se é redigido apenas o último, sendo que eles podem ser combinados sempre que for apropriado (BOURQUE e FAIRLEY, 2014).

Especificação de Requisitos de *Software*

Especificação de requisitos de *software* estabelece a base para um acordo entre clientes e prestadores de serviços ou fornecedores, definindo o que o *software* desenvolvido deve fazer ou não. É uma ferramenta importante para avaliar as necessidades do cliente, serve como base para estimar os custos do produto, riscos, desenvolver planos de verificação e validação eficazes.

Requisitos de *software* são muitas vezes escritos em linguagem natural, que podem ser complementadas por descrições formais ou semiformais. Algumas notações apropriadas permitem que requisitos específicos e aspectos arquiteturais sejam descritas de forma mais concisa do que "na linguagem natural". É importante que os requisitos sejam bem escritos, para evitar problemas de interpretação pelos membros da equipe desenvolvedora e seus clientes, garantindo um *software* que atinja melhor as necessidades dos usuários (BOURQUE e FAIRLEY, 2014).

1.1.3. Projeto de *Software*

O projeto de *software* é o momento onde os requisitos são analisados a fim de produzir uma descrição da estrutura interna do sistema que será desenvolvido (BOURQUE e FAIRLEY, 2014).

Existem muitas notações para representar artefatos de projeto de *software*, a fim de descrever a organização estrutural de um projeto ou o comportamento do sistema, tais como diagramas de classes e conceitos, diagrama de componentes, diagramas de entidade-relacionamento (usado para representar modelos conceituais do armazenamento nos bancos de dados), diagramas de estrutura (descreve a estrutura interna do *software*) (BOURQUE e FAIRLEY, 2014).

1.1.4. Modelagem Conceitual

A finalidade de se desenvolver modelos é fundamental para análise de requisitos do *software*, pois ajuda a compreensão da situação e a solução para os questionamentos. Assim, modelos conceituais compreendem modelos de entidades de domínio do problema, para refletir a atual situação e suas dependências. Para ilustrar estes modelos, pode-se usar diagramas de casos de uso, modelos de fluxo de dados, modelos baseados em metas entre outros (BOURQUE e FAIRLEY, 2014). Muitas destas notações são feitas usando UML (*Unified Modeling Language*), uma linguagem de modelagem projetada para especificar, visualizar e documentar modelos de sistemas de *software*, incluindo a sua estrutura e design, de forma padronizada (OBJECT MANAGEMENT GROUP, 2014).

1.1.5. Projeto Arquitetural

Segundo BOURQUE e FAIRLEY uma arquitetura de *software* poderia ser definida como o conjunto de estruturas necessárias para o funcionamento do sistema, compreendendo elementos de *software*, as relações entre eles e suas propriedades. Esta definição a cada dia vem sendo mais abrangente, compreendendo alguns conceitos como estilos arquitetônicos, arquitetura de banco de dados, padrões de projeto), etc.

1.2. Banco de dados

Bancos de dados são "conjuntos de dados integrados que têm por objetivo atender a uma comunidade de usuários" (HEUSER, 1998). Segundo Date (2004), um banco de dados seria o equivalente eletrônico a um armário de arquivamento que serve de repositório para uma coleção de dados. Para Rob e Coronel (2011), "dados" são fatos brutos, ou seja, não foram processados para revelar seu significado. Este processamento pode ser simples ou complexo e dependem de um contexto. Após a realização do processamento dos dados obtêm-se informações que podem ser utilizadas como fundamento para tomada de decisões.

1.2.1. Tipos de Banco de Dados

Existem vários modelos de banco de dados disponíveis no mercado atualmente, os mais conhecidos são:

- Banco de Dados Hierárquico: O primeiro dos modelos a ser reconhecido como banco de dados onde "os dados são estruturados em hierarquias ou árvores. Os nós das hierarquias contêm ocorrências de registros, onde cada registro é uma coleção de campos (atributos), cada um contendo apenas uma informação" (TAKAI, ITALIANO e FERREIRA, 2005). Um exemplo desse tipo de banco de dados é o *Information Management System* (IMS) da IBM (MACÁRIO e BALDO, 2005).
- Banco de Dados Relacional: um conjunto de tabelas com nomes únicos compõem o banco de dados, e pode existir a relação entre uma ou mais tabelas na base (DATE, 2004) (BOSCARIOLI, BEZERRA, *et al.*, 2006). Neste modelo algumas restrições precisam ser levadas em consideração para evitar repetição de informação, a perda e/ou a incapacidade de representar parte da informação: integridade referencial, chaves e integridade de junções de relações (TAKAI, ITALIANO e FERREIRA, 2005). Um exemplo é o MySQL (MYSQL)

- Banco de Dados Orientado a Objetos: Este tipo de banco de dados realiza o armazenamento de dados na forma de objetos, bem como estruturas de dados utilizadas, mantendo suas características e funções particulares (TAKAI, ITALIANO e FERREIRA, 2005) (GOMES, 2009). Exemplos de SGBD Orientados a objetos são o Versant (ACTIAN) (DATE, 2004) e o Caché (INTERSYSTEMS).
- Banco de Dados Objeto-Relacional: Esse tipo de banco de dados tenta suprir as dificuldade de representar dados complexos existente nos sistemas relacionais e adicionar as facilidades para manusear esse tipo de dados usando SQL (*Structured Query Language*) (TAKAI, ITALIANO e FERREIRA, 2005). Um exemplo de SGBD objeto-relacional de código aberto é o PostgreSQL (POSTGRESQL).

1.2.2. Linguagem de Consulta Estruturada

Considerado por Navathe e Elmasri (2011) como um dos principais motivos do sucesso dos bancos de dados relacionais comerciais, a *Structured Query Language*, ou "Linguagem de Consulta Estruturada", conhecida como SQL, é a linguagem padrão para se lidar com banco de dados relacionais desenvolvida na década de 1970 por um grupo de pesquisas da IBM (DATE, 2004). A SQL inclui operações de definição (como o comando **CREATE**) e manipulação de dados (**SELECT**, **INSERT**, **UPDATE** e **DELETE**).

1.2.3. Sistemas de Gerenciamento de Banco de Dados

Para manipular esses dados foram desenvolvidos os chamados "Sistemas de Gerenciamento de Banco de Dados" - SGBD. Um SGBD é um *software* que incorpora as funções de definição, recuperação e alteração de dados em um banco de dados (HEUSER, 1998). Os SGBDs fazem parte do Ambiente de Banco de dados, composto por: *software* (aplicações e SGBD), dados, *hardware* (HD – *Hard Disk*) e usuários (DATE, 2004) (GOMES, 2009).

Espera-se que um SGBD tenha os seguintes aspectos (GOMES, 2009):

- Flexibilidade: Permitir a criação de novas bases de dados e a especificação dos seus esquemas;
- Abstração: é possível realizar consultas e modificações nos dados usando uma linguagem apropriada chamada "linguagem de consulta" ou "linguagem de manipulação de dados";
- Durabilidade: Manter dados armazenados por um longo período de tempo, de forma segura;
- Concorrência: Realizar o controle de acesso de múltiplos usuários simultaneamente, sem permitir que estes acessos afetem uns aos outros e que os dados sejam corrompidos.

1.2.4. Sistema de Gerenciamento de Banco de Dados MySQL

O MySQL é um sistema de gerenciamento de banco de dados relacional multitarefa e multiusuário de código aberto, ou seja, é possível usar e modificar o *software* (MYSQL). Foi criado pelos suecos David Axmark e Allan Larsson e o finlandês Michael Widenius, e em 2008 foi adquirido pela Sun Microsystems (DINIZ, 2011).

O MySQL apresenta "portabilidade, alta compatibilidade de drivers, módulos de interface para diversas linguagens de programação, suporta Unicode (padrão que permite representar e manipular texto de diversos sistemas de escrita), *Full Text Indexes*, replicação, *Hot Backups*, GIS, OLAP" (DINIZ, 2011). É um SGBD popular principalmente para aplicações *web*.

Engines

Engines são componentes do MySQL que lidam com as operações SQL para diferentes tipos de tabelas. Utilizando o comando SHOW ENGINE é possível verificar quais mecanismos de armazenamento são suportados pelo servidor.

As *engines* suportadas pelo MySQL 5.1, versão utilizada atualmente pela InovaBit, são (MYSQL):

- **InnoDB:** Foi desenvolvida para o processamento de transações de alta confiabilidade e desempenho, que segue o modelo ACID (Atomicidade, Consistência, Isolamento e Durabilidade, do inglês: *Atomicity, Consistency, Isolation, Durability*) com operações como COMMIT (fazer um conjunto de mudanças permanente) e ROLLBACK (ignorar as alterações realizadas desde o início de uma transação). Além disso, o InnoDB permite o bloqueio em nível de linha, o que permite aumentar a simultaneidade de usuários, e o uso de chaves estrangeiras (FOREIGN KEY) para restrições referenciais de integridade (MYSQL) (SCHWARTZ, ZAITSEV e TKACHENKO, 2012).
- **MyISAM:** É uma *engine* baseada no ISAM (Método de Acesso Sequencial Indexado, do inglês *Indexed Sequential Access Method*). O MyISAM oferece funcionalidades de indexação de texto, compressão e funções espaciais, e não suporta transações ou bloqueios a nível de linha. (SCHWARTZ, ZAITSEV e TKACHENKO, 2012).
- **Memory:** Anteriormente conhecida como HEAP, é uma *engine* que armazena todos os dados na memória RAM, o que significa que eles estão vulneráveis a quedas de energias ou outros tipos de eventos que interrompam o funcionamento do servidor. Os bloqueios são em nível de tabela durante as consultas, o que dá uma concorrência de escrita muito baixa (SCHWARTZ, ZAITSEV e TKACHENKO, 2012). Além disso, devido ao fato das tabelas serem armazenadas na memória RAM, não é uma boa opção para tabelas muito grandes, o armazenamento tem tamanho fixo, ou seja, tipos com tamanho variável como "VARCHAR" são armazenados no seu tamanho máximo, independente de ter sido informado um tamanho menor durante a criação da tabela. (CURIOSO, BRADFORD e GALBRAITH, 2010) (MYSQL). É comum confundir as tabelas *Memory* com tabelas temporárias (criadas com CREATE TEMPORARY TABLE). As tabelas temporárias podem usar qualquer *engine* e, diferentemente das tabelas *Memory*, as tabelas temporárias são visíveis somente à única conexão e desaparecem completamente quando a conexão é encerrada (SCHWARTZ, ZAITSEV e TKACHENKO, 2012).

- **Merge:** também conhecida como MRG_MyISAM, é um conjunto de tabelas MyISAM idênticas (possui colunas e índices idênticos) que podem ser usadas como uma, desde que não tenham ordem diferentes, colunas diferentes ou índices em ordem diferente (CURIOSO, BRADFORD e GALBRAITH, 2010).
- **Archive:** As tabelas criadas com a *engine Archive* são compactas e sem índices, destinadas a armazenar grandes quantidades de informação raramente utilizadas. Esta *engine* suporta operações de inserção e seleção, mas não suporta operações de remoção ou alterações.
- **Federated:** Oferece a capacidade de vincular servidores MySQL separados para criar um banco de dados lógico de muitos servidores físicos. É possível criar uma tabela que é uma representação local de uma tabela remota (MYSQL). Usando a *engine Federated* os detalhes dos dados, como a forma de armazenamento, não são conhecidos pelo usuário, pois a fonte de dados possui autonomia. Desta mesma forma esta *engine* não permite modificar na fonte a definição de dados usando declarações como "ALTER TABLE" (CURIOSO, BRADFORD e GALBRAITH, 2010).
- **CSV:** Esta *engine* armazena dados em arquivos de texto usando o formato CSV, que pode ser lido por diversos sistemas de edição de planilhas, como o Microsoft Excel ou LibreOffice Calc. (MYSQL) (SCHWARTZ, ZAITSEV e TKACHENKO, 2012).
- **Blackhole:** Age como um "buraco negro" (do inglês "*black hole*") que aceita dados, mas não os armazena. Se o log binário estiver habilitado, as instruções SQL enviadas são registradas e podem ser replicadas em servidores *slaves*, o que pode ser útil em técnicas de replicação e segurança (CURIOSO, BRADFORD e GALBRAITH, 2010).
- **Example:** Permite criar tabelas onde dados não são inseridos (e por isso, não podem ser recuperados). A finalidade desta *engine* é servir como um exemplo no código fonte do MySQL como base para o desenvolvimento de novas *engines* (MYSQL).

1.2.4.1. Otimização em Banco de Dados MySQL

O desempenho do banco de dados depende de vários fatores no nível de banco de dados, como tabelas, consultas e definições de configuração. É possível trabalhar para que as operações sejam mais eficientes. É preciso verificar se o *hardware* está satisfazendo as necessidades, se as tabelas estão estruturadas corretamente, se os índices estão criados de forma correta à otimizar buscas, se a *engine* escolhida é a melhor para a situação e se as configurações de *cache* estão satisfatórias, ou seja, grande o suficiente para armazenar dados acessados com frequência, mas não tão grande que eles sobrecarregam a memória RAM) (MYSQL).

1.2.4.2. Otimização de consultas SQL

As atividades lógicas de um SGBD são realizadas através de instruções SQL. É importante melhorar as instruções SQL visando melhoria de performance nas aplicações que utilizam o banco de dados.

Segundo DATE (2004), existem quatro grandes estágios de consultas SQL:

- 1) Converter a consulta para algum formato interno: este estágio envolve a conversão da consulta original em uma representação mais adequada à manipulação pelo servidor;
- 2) Converter para a forma canônica: são executadas uma série de otimizações, visando encontrar uma representação mais eficiente do que a original. Uma busca pode ser escrita de diversas formas, e o desempenho de uma consulta não pode depender exclusivamente do modo em que o usuário escolheu escrevê-la;
- 3) Escolher procedimentos candidatos de baixo nível: nesta fase é considerada a existência de índices ou outros caminhos de acesso, distribuição de valores dos dados, a parte física de armazenamento de dados.
- 4) Gerar planos de consultas e escolher o mais barato: São gerados planos de consulta e escolhe-se o mais econômico. A avaliação de custos é difícil, em consultas mais longas há a geração de resultados intermediários em tempo de execução, além da possibilidade combinatória de consultas poder ser muito grande, se fazendo necessário algum tipo de heurística

para manter o conjunto gerado dentro dos limites, ou seja, reduzir o espaço de pesquisa.

O SELECT é uma declaração SQL que permite recuperar zero ou mais linhas dos dados de um objeto de banco de dados, como uma tabela ou *view*. Para melhorar instruções usando SELECT uma das primeiras ações é verificar em quais campos devem-se criar índices. Os índices ajudam consultas que fazem referencia a tabelas diferentes usando *joins* (cláusula usada para combinar duas ou mais tabelas) e chaves estrangeiras (campo que estabelece o relacionamento entre duas tabelas). Para auxiliar na escolha dos campos que devem ser índices é possível utilizar o comando EXPLAIN, que mostra quais índices são utilizados em uma instrução (MYSQL).

Algumas medidas podem ser tomadas também durante a declaração das condições usando WHERE, como a utilização de campos indexados para otimizar as buscas, a remoção de parêntesis excessivos, utilizar o comando WHERE em uniões de tabelas para diminuir a quantidade de linhas a serem analisadas,

Índices

Índices são estruturas de acesso que representam a prioridade de um registro, são usados para agilizar a recuperação de registro em resposta a certas conduções de pesquisa (NAVATHE e ELMASRI, 2011). Os índices são geralmente utilizados para encontrar registros em clausulas WHERE, em junção de tabelas na clausula ON, para encontrar o valor máximo MAX() e mínimo MIN(), para ordenar ou agrupar tabelas (MYSQL) (FERREIRA, 2009). Sem índice o SGBD precisará percorrer por toda a tabela até que sejam encontrados os registros com relevância, ou seja, quanto maior a tabela, maior o custo.

Apesar de aumentar a velocidade de consultas, os índices diminuem o desempenho das operações de escrita (inserções, remoções e atualizações) (FERREIRA, 2009) (SCHWARTZ, ZAITSEV e TKACHENKO, 2012). DINIZ (2011) conclui que é importante saber criar e usar índices corretamente para o bom desempenho da consulta.

Todos os tipos de dados no MySQL podem ser indexados, mas os índices em colunas relevantes são os que auxiliam na performance de operações SELECT. Os

mecanismos suportam até 16 índices por tabela, sendo que há a possibilidade de criar índices múltiplos de no máximo 16 colunas cada. Um índice múltiplo é como um *array* ordenado de valores concatenando os dados das colunas indexadas (MYSQL).

Os índices *B-Tree* utilizados pela maior parte das *engines* do MySQL podem ser usados em comparações usando os operadores = (igual), > (maior), >= (maior ou igual), < (menor), <= (menor ou igual), ou BETWEEN. Se for realizado o uso do operador LIKE apenas será considerado o índice se o argumento para este operador for uma *string* constante que não inicia com caractere coringa %. É recomendado pelo manual do MySQL a criação de índices em colunas usadas na clausula WHERE de instruções SQL para acelerar a avaliação, filtragem e a recuperação final dos resultados.

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

Figura 1: Exemplo de declaração SQL que não utiliza índice. Na primeira declaração usa-se %, na segunda a *string* de comparação não é constante (MYSQL).

Log de Consultas

O MySQL possui em sua configuração dois tipos de *logs* que podem ser ativados para registrar as consultas realizadas: o log geral, que contém todas as consultas recebidas pelo servidor, e o *log* de consultas lentas, que contém todas as consultas que demoraram determinado tempo para serem executadas. É possível utilizar ambos para análise de desempenho, contudo o *log* de consultas lentas já registrará as consultas problemáticas (SCHWARTZ, ZAITSEV e TKACHENKO, 2012).

Para cada consulta gravada no *log* de consultas lentas existem três linhas com informações sobre a execução, conforme ilustrado na Figura 2:

1. Data em que a consulta foi realizada;
2. Usuário do banco de dados que executou a consulta;
3. O tempo em segundos que a consulta demorou a ser executada (*Query_time*), o tempo em que a consulta precisou esperar por bloqueios na tabela (*Lock_time*) e a quantidade de linhas examinadas para se obter o resultado desejado (*Rows_examined*).

```

1 # Time: 111007 19:03:07
2 # User@Host: root[root] @ localhost []
3 # Query_time: 8 Lock_time: 0 Rows_sent: 500 Rows_examined: 111206252
4 SELECT a.ProdutoCod ...

```

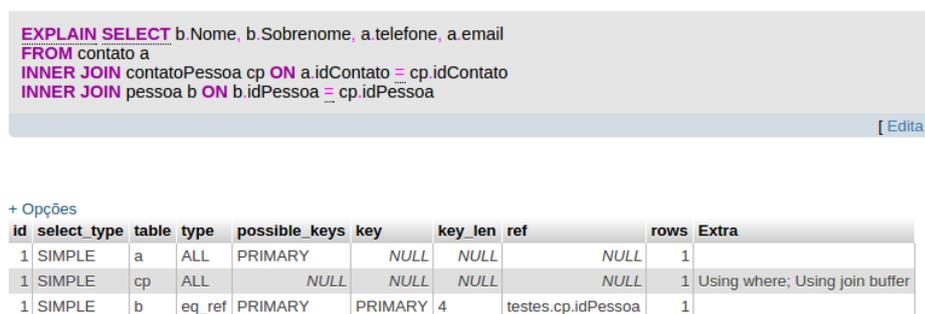
Figura 2: Arquivo de log de consultas lentas (DINIZ, 2011).

Plano de execução de *Query*

Dependendo das colunas, índices e condições na cláusula WHERE de uma *query* o MySQL realiza diversas operações para melhorar a performance e eficiência do comando executado. Uma declaração em uma tabela grande pode ser executada sem precisar percorrer todas as colunas fazendo comparações de linhas. Para verificar as operações realizadas pelo otimizador nativo do MySQL a fim de melhorar a eficiência da *query* é possível exibir o "Plano de execução de *Query*" usando o comando EXPLAIN (MYSQL).

O comando EXPLAIN exibe informações como a ordem de junção de tabelas, as chaves possíveis e as linhas percorridas. É relevante ressaltar que EXPLAIN não fornece informações como gatilhos, *stored functions*, não funciona para *stored procedures* (apenas se executado o comando EXPLAIN separadamente para cada um) e algumas estimativas podem ser imprecisas (MYSQL).

Para executar este comando basta adicionar a palavra EXPLAIN antes de SELECT em uma cláusula SQL.



```

EXPLAIN SELECT b.Nome, b.Sobrenome, a.telefone, a.email
FROM contato a
INNER JOIN contatoPessoa cp ON a.idContato = cp.idContato
INNER JOIN pessoa b ON b.idPessoa = cp.idPessoa

```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	a	ALL	PRIMARY	NULL	NULL	NULL	1	
1	SIMPLE	cp	ALL	NULL	NULL	NULL	NULL	1	Using where; Using join buffer
1	SIMPLE	b	eq_ref	PRIMARY	PRIMARY	4	testes.cp.idPessoa	1	

Figura 3: Exemplo de utilização da cláusula EXPLAIN no phpMyAdmin.

Schwartz, Zaitsev e Tkachenko (2012) explicam as colunas resultantes da cláusula EXPLAIN, ressaltando que a ordem em que são exibidas as linhas é a ordem que o MySQL executa as partes da consulta e não a ordem no *script* SQL enviado:

- Coluna **id**: contém o número que identifica o SELECT. Caso não haja sub consultas ou UNIONS na declaração apenas existirá um SELECT. Caso contrário, as instruções internas serão enumeradas sequencialmente de acordo com suas posições no comando original.
- Coluna **select_type**: Mostra a complexidade da instrução. O valor SIMPLES significa que a consulta não contém sub consultas ou UNIONS. Se a consulta tem quaisquer sub consultas complexas, a parte mais externa é chamada PRIMARY e outras partes são denominadas:
 - **SUBQUERY**: Um SELECT que está contido em uma sub consulta de itens a serem "selecionados"
 - **DERIVED**: Um SELECT na cláusula FROM. O MySQL executa de forma recursiva e armazena em uma tabela temporária.
 - **UNION**: a partir da segunda tabela em comandos com UNION.
 - **UNION RESULT**: O comando SELECT usado para recuperar resultados de UNIONS anônimos na tabela. Além disso, SUBQUERY e UNION podem ser denominadas como DEPENDENT (significa que dado comando SELECT depende de dados que são encontrados em outra consulta) ou UNCACHEABLE (algo no SELECT impede que os resultados sejam armazenados em cache).
- Coluna **table**: Exibe as tabelas acessadas pela consulta.
- Coluna **type**: Exibe o tipo de acesso que a consulta realizou. Esta informação pode ser:
 - **ALL**: significa que o MySQL precisará percorrer toda a tabela, do início ao fim, para achar a linha desejada;
 - **index**: Assim como o acesso ALL, este tipo de acesso percorre toda a tabela, porém ordenada pelos índices. Isto significa que a busca será feita de forma randômica, o que pode aumentar o custo de execução;
 - **range**: ocorre em consultas com BETWEEN, IN ou ON na cláusula WHERE. Não faz uma busca completa na tabela, inicia

em algum ponto e retorna as linhas que satisfazem o intervalo desejado.

- **ref**: neste tipo de acesso o índice é comparado a um valor de referência, que pode ser uma constante ou o resultado de uma consulta anterior (sub consultas).
- **eq_ref**: busca de um índice que o MySQL sabe que terá um único valor como retorno, ou seja, uma chave primária ou um índice do tipo UNIQUE.
- **const, system**: este é o tipo de acesso onde o MySQL otimizou algum acesso na consulta. Por exemplo, quando uma chave primária for inserida na cláusula WHERE, o MySQL a insere como constante para otimizar os tempo de resposta.
- **NULL**: Significa que o MySQL sequer acessou a tabela que estava no comando SELECT inicial, como por exemplo, o SELECT do valor mínimo a partir de um índice pode ser feito analisando apenas o índice, e não requer o acesso a tabela durante a execução.
- Coluna **possible_keys**: Lista os índices que podem ser utilizados para otimizar a consulta.
- Coluna **key**: O índice que o MySQL usou para acessar a tabela
- Coluna **key_len**: Mostra o número de bytes que o MySQL usará no índice.
- Coluna **ref**: Mostra quais as colunas de tabelas anteriores estão sendo usadas para procurar valores no índice nomeados na coluna **key**.
- Coluna **rows**: Exibe o número de linhas que o MySQL estima que terá de percorrer até satisfazer os critérios da consulta.
- Coluna **extra**: Esta coluna contém informações extras que não se encaixam em outras colunas. As informações que mais comumente são inseridas nesta coluna são:
 - **Using index**: indica que o MySQL irá usar um índice para evitar acesso à tabela;

- ***Using where***: depois da recuperação das linhas pela ferramenta de armazenamento, o MySQL irá filtrá-las;
- ***Using Temporary***: indica o uso de uma tabela temporária;
- ***Using filesort***: significa que o MySQL irá ordenar os resultados com uma classificação externa.

2. MATERIAIS, TÉCNICAS E MÉTODOS

Neste capítulo serão descritos os materiais, técnicas e métodos utilizados para o desenvolvimento das atividades realizadas durante a realização do estágio supervisionado.

2.1. Materiais

Para a realização deste trabalho foi utilizado um computador com o sistema operacional Ubuntu, e os *softwares* utilizados durante o estágio são os descritos a seguir:

- **yEd Graph Editor:** programa de diagramação que suporta diversos tipos de diagramas, como UML, mapas mentais, diagrama de entidade-relacionamento, entre outros. É compatível com qualquer sistema operacional que suporte JVM. O yEd foi utilizado neste trabalho para o desenho dos casos de uso e diagramas de conceitos.
- **MySQL:** Gerenciador de Banco de Dados na versão 5.1 no ambiente de teste e de produção.
- **PhpMyAdmin:** Sistema que fornece interface gráfica para administração do SGBD MySQL.
- **NetBeans:** Ambiente de desenvolvimento compatível com Windows, Linux, e MacOS, que fornece ferramentas necessárias para o desenvolvimento de sistemas. Foi utilizado para manipulação e edição dos arquivos de códigos do sistema.
- **Neor Profile SQL:** ferramenta gráfica gratuita de monitoramento de *queries* em banco de dados MySQL em sistemas operacionais Windows, MacOS e Linux. Este programa foi utilizado para testes de *queries* SQL.

2.2. Técnicas e Métodos

As atividades do estágio supervisionado foram realizadas seguindo a seguinte estruturação:

- Estudo de bibliografia: Esta etapa ocorreu em paralelo com as demais etapas durante todo o estágio. Foram realizados estudos que se julgaram necessários para o desenvolvimento do trabalho, sendo os principais pontos previamente abordados no primeiro capítulo deste documento.
- Estudo do *framework* InovaBit: Estudo para compreensão da arquitetura do sistema, das principais classes utilizadas e dos padrões de código definidos pela equipe de desenvolvedores da empresa.
- Estudo sobre o sistema GeAgroBit: Compreensão do fluxo dos processos do sistema, análise dos módulos "Contas a Pagar e Receber" e "Pagamento de Frete", com o desenvolvimento de casos de uso e diagrama de conceito para melhor compreensão dos módulos e para anexação e atualização da documentação previamente existente.
- Desenvolvimento de uma solução para o problema proposto: Análise do problema apontado pelo cliente, levantamento e testes da situação atual, elaboração e desenvolvimento de solução para o problema verificado.

Os resultados obtidos em todas estas etapas são descritos no capítulo a seguir.

3. ESTUDO DE CASO

Neste capítulo serão descritos todos os resultados obtidos ao longo do período de estágio supervisionado, sendo estes relacionados à otimização dos módulos "Pagamento de Frete" e "Contas a Pagar e Receber" do sistema GeAgroBit.

3.1. *Framework InovaBit*

Um *framework* é um conjunto de classes abstratas ou concretas, visando à reutilização de código e serve como estrutura a ser ampliada para criar um sistema mais específico (SOMMERVILLE, 2007).

A InovaBit criou um *framework* utilizando a linguagem de programação PHP para agrupar classes comuns a todos os sistemas desenvolvidos pela empresa. Existe uma documentação para auxílio do uso das classes e funções, contudo esta se encontra incompleta e muitas atualizações do *framework* não se estão devidamente documentadas. Isto gera certa dificuldade aos novos colaboradores da empresa, fazendo com que, ao invés de utilizar a documentação, usem exemplos de código em outros arquivos desenvolvidos pela equipe e tentem entender os funcionamentos pela tentativa e erro.

Os sistemas que utilizam o *framework* InovaBit possuem uma estrutura básica nos arquivos de seus módulos, ilustradas na Figura 4.e descritas abaixo, onde a palavra "modulo" é o nome de arquivo variável por módulo:

modulo.php: Este arquivo é o primeiro arquivo acessado. Ele agrupa os demais arquivos de forma a ser uma visualização ao usuário. Faz chamada das classes do *framework* a serem utilizadas, contendo os botões, o filtro, o contêiner e a grid com as informações do módulo;

modulo.form.php: possui as definições dos campos de filtro e dos formulários a serem usados no módulo. É neste arquivo também que estão os códigos em *javascript*;

modulo.ajax.php: é responsável por dar retorno as requisições em AJAX chamadas através do arquivo modulo.form.php.

modulo.class.php: é a classe principal do módulo. Possui as funcionalidades que controlam e modelam as informações buscadas nos bancos de dados. Todo arquivo `modulo.class.php` dos módulos estender o arquivo `modulo.sql.php`.

modulo.sql.php: este arquivo é responsável pelo retorno de *queries* SQL para serem utilizadas nas classes. Não realiza diretamente a conexão com banco de dados, esta é tarefa do arquivo `modulo.class.php`.

modulo.tpl.php: responsável pela formatação da visualização dos formulários.

modulo.print.php: responsável por gerar a visualização do arquivo a ser impresso pela função do *framework*.

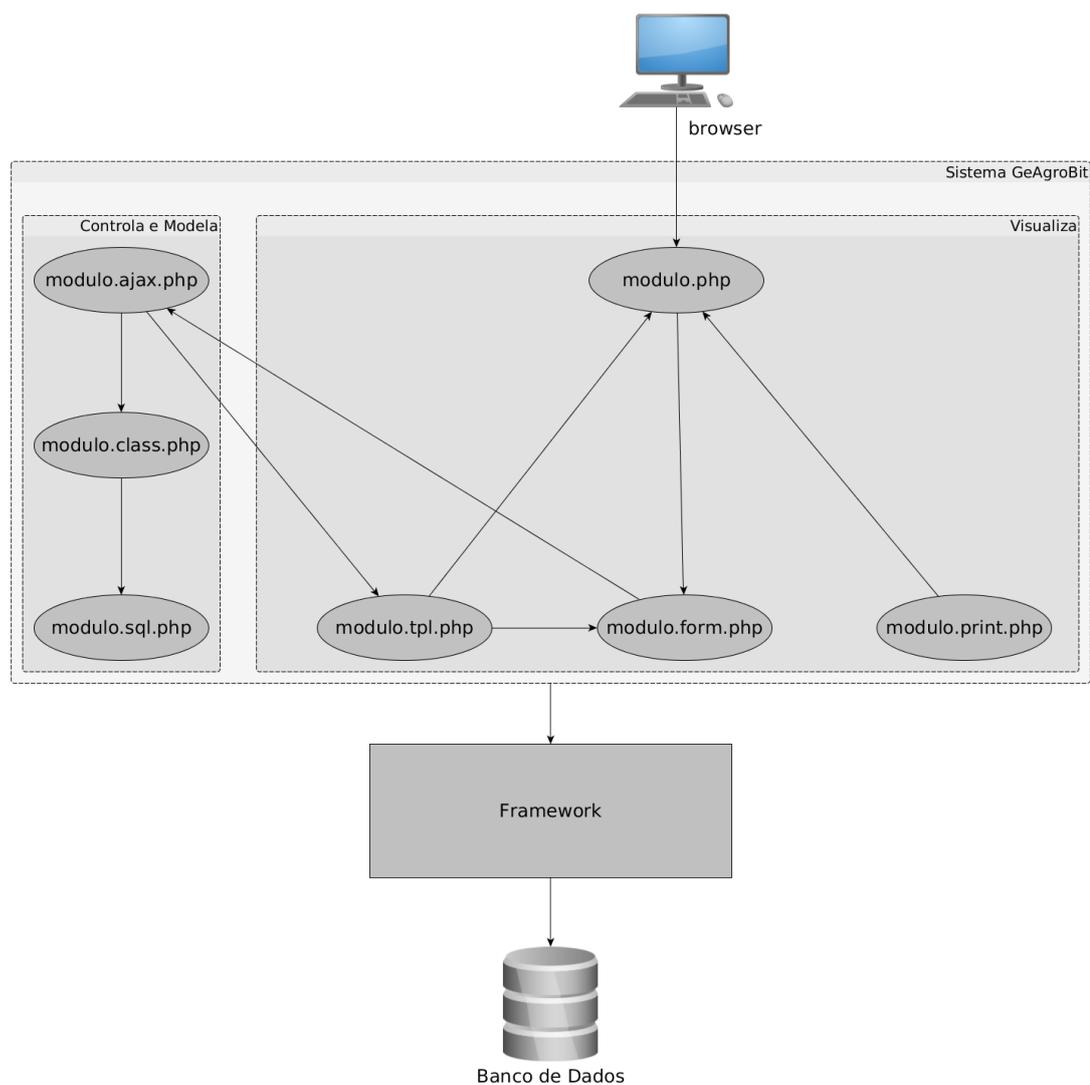


Figura 4: Arquitetura dos sistemas que utilizam o *framework* InovaBit.

3.1.1. Classe Monitor

O *framework* InovaBit possui uma classe para monitoramento de desempenho de código chamada "Monitor", ilustrada na Figura 5 e com código disponível no ANEXO 1. O código desenvolvido utiliza a função nativa do PHP *microtime*, que retorna um *timestamp* (valor em que determinado evento ocorreu) (PHP GROUP)). A classe Monitor possui a função *inicioMonitoramento()* que armazena o início do monitoramento em uma variável de classe e realiza uma inserção de dados na tabela *_monitor* no banco de dados com o tempo de monitoramento de valor nulo. Ao final do código que se deseja monitorar é preciso chamar a função *fimMonitoramento()*, que calcula o tempo de execução e chama a função *salvaMonitoramento()* que atualiza a tabela *_monitor* no banco de dados com o tempo de execução calculado.

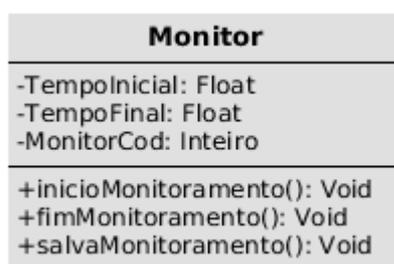


Figura 5: Classe Monitor do *framework* InovaBit.

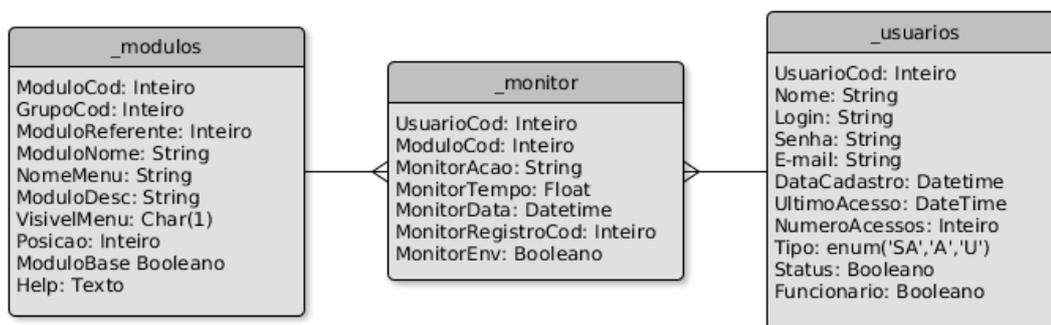


Figura 6: Tabelas no banco de dados mencionadas no módulo Monitor.

3.1.2. Filtro de Informações

Os módulos dos sistemas que utilizam o *framework* InovaBit possuem campos para serem utilizados como filtro de informações desejadas, conforme

ilustrado na Figura 7. Através dele os usuários costumam a buscar informações dos dados disponibilizados no módulo, facilitando a busca de informações relevantes.

Figura 7: Trecho do filtro do módulo "Contas a pagar e Receber" sistema GeAgroBit.

O sistema disponibiliza opções de busca diferentes ao clicar no ícone de sinal = ao lado do campo desejado, possibilitando a busca por período de tempo ou parte de textos. As notações estão descritas no Quadro 1 e impactam diretamente no código SQL que realizará a busca das informações utilizando a classe Filtrar.

Quadro 1: Significado dos botões nos campos do filtro no sistema GeAgroBit

Opção de botão do filtro	Significado
=	Permitirá buscar os registros onde a informação será IGUAL ao do campo preenchido no filtro.
<>	Permitirá buscar os registros onde a informação será DIFERENTE do campo preenchido no filtro
>	Permitirá buscar os registros onde a informação será MAIOR do campo preenchido no filtro
<	Permitirá buscar os registros onde a informação será MENOR do campo preenchido no filtro
>=	Permitirá buscar os registros onde a informação será MAIOR OU IGUAL do campo preenchido no filtro
<=	Permitirá buscar os registros onde a informação será MENOR OU IGUAL do campo preenchido no filtro
E	Permitirá buscar os registros onde a informação vai satisfazer aos dois campos preenchidos. Muito útil para buscas usando intervalo de tempo. Esta opção abrirá um espaço na tela onde poderão ser preenchidos dois campos.
OU	Permitirá buscar os registros onde a informação é igual a um dos dois campos preenchidos. Esta opção abrirá um espaço na tela onde poderão ser preenchidos dois campos
*	Buscará registros onde a informação contém o texto digitado preenchido no filtro.
*A	Buscará registros onde a informação termina com o texto digitado preenchido no filtro.
A*	Buscará registros onde a informação inicia-se com o texto digitado preenchido no filtro.

Fonte: Página de documentação do sistema GeAgroBit.

3.1.3. Classe Filtrar

A classe Filtrar está disponível no *framework* InovaBit e é utilizada para realizar a adição dinâmica das cláusulas WHERE nas funções SQL. A função mais utilizada da classe é "getStringSql", que recebe como parâmetros o nome do campo do formulário definido no arquivo "*.form", o nome do campo no banco de dados referente a informação buscada e o tipo de informação contida no campo (texto, data, inteiro, *float*). Esta função retorna texto em formato de código SQL levando em consideração também o operador do filtro selecionado (Quadro 1). Normalmente a função "getStringSql" é chamada no fim da função que retorna o código SQL completo, conforme exemplo na Figura 8, retirado do módulo de cadastro de bancos do sistema GeAgroBit:

```

6      public function filtrarSql($ObjForm)
7      {
8          //Filtro Dinamico
9          $Fil = new Filtrar($ObjForm);
10
11         $Sql = "SELECT BancoCod, BancoCodigo, BancoNome,
12                CASE
13                WHEN BancoSituacao = 'A' THEN 'Ativo'
14                WHEN BancoSituacao = 'I' THEN 'Inativo'
15                END BancoSituacao
16                FROM banco
17                WHERE 1 ";
18
19         $Sql .= $Fil->getStringSql("BancoNome", "BancoNome", "Texto");
20         $Sql .= $Fil->getStringSql("BancoSituacao", "BancoSituacao", "Texto");
21         $Sql .= $Fil->getStringSql("BancoCodigo", "BancoCodigo", "Texto");
22         //Sql de Impressão
23         $Sql .= $Fil->printSql("BancoCod", $_GET['SisReg']);
24
25
26         return $Sql;
27     }

```

Figura 8: Código da função "filtrarSql" do módulo Bancos do sistema GeAgroBit, que utiliza a classe Filtrar do *framework*.

Seguindo o exemplo acima, a busca por um banco utilizando a opção asterisco no botão do campo "Nome Banco", de situação Ativa e de código "001" no filtro resulta no seguinte código SQL:

ocultar filtros mostrar filtros salvos salvar filtro

Código do Banco: = 001 Nome do Banco: * Brasil

Situação: = Ativo

Filtrar Limpar

adicionar editar visualizar excluir atualizar exportar csv inicial ajuda

imprimir sair

```
SELECT BancoCod, BancoCodigo, BancoNome,
CASE
WHEN BancoSituacao = 'A' THEN 'Ativo'
WHEN BancoSituacao = 'I' THEN 'Inativo'
END BancoSituacao
FROM banco
WHERE 1 AND BancoNome LIKE '%Brasil%' AND BancoSituacao = 'A' AND BancoCodigo = '001'
```

Figura 9: Retorno da função "filtrarSql" do módulo Banco do sistema GeAgroBit.

3.2. O sistema GeAgroBit

A InovaBit iniciou em 2010 o desenvolvimento de um sistema *web* de gestão agrícola voltado para uma fazenda de grande porte. Desenvolvido em linguagem PHP e banco de dados MySQL, possuía inicialmente poucos módulos e vem agregando novas funcionalidades continuamente desde então.



Figura 10: Tela inicial do Sistema GeAgroBit após realização de *login*.

Atualmente o sistema não possui uma documentação consistente, o que leva novos e antigos desenvolvedores da empresa a encontrarem dificuldades para compreender o funcionamento do sistema durante o desenvolvimento de novas funcionalidades ou manutenção de código. Além disso, existem alguns problemas de projeto, como a falta de uma solução para o aumento da quantidade de dados que o sistema necessita manipular, o que gera a lentidão em alguns módulos.

O sistema possui seus processos definidos conforme a Figura 11:

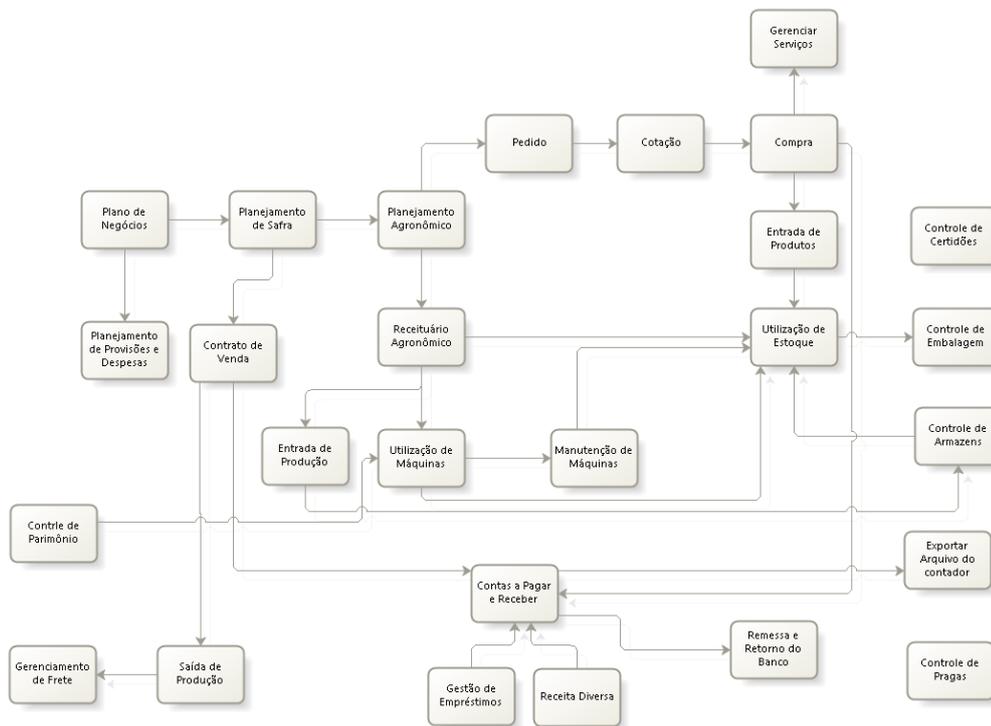


Figura 11: Fluxo de processos do sistema GeAgroBit.

O banco de dados de produção do sistema GeAgroBit é o MySQL 5.1, e a *engine* utilizada como padrão no sistema é a InnoDB, pois permite o trabalho com transações e bloqueio em nível de linha. Os desenvolvedores do sistema optaram não definir as chaves estrangeiras via banco de dados, sendo esse tipo de relação e suas validações tratadas diretamente na aplicação.

3.2.1. Definição do problema de performance

Foram registradas diversas reclamações dos clientes em relação à demora do carregamento da página ao abrir o módulo ou ao carregar informações utilizando os filtros em "Pagamento de Frete" e "Contas a Pagar e Receber". Os usuários alegavam que o tempo para buscar as informações estava muito alto e atrapalhando o fluxo de trabalho. Para verificar a situação foram realizados testes de desempenho a fim de identificar o problema.

3.3. Módulos "Pagamento de Frete" e "Contas a Pagar e Receber"

Foi realizada a descrição dos casos de uso dos módulos para o entendimento das funcionalidades existentes e compreensão da origem das informações manipuladas. Estes artefatos foram também adicionados à documentação do sistema. Depois fora realizado um levantamento para averiguar as causas do mau desempenho e por fim o desenvolvimento de uma solução.

3.3.1. Módulo "Contas a Pagar e Receber"

O módulo "Contas a Pagar e Receber" cadastra a quitação de parcelas a pagar e receber geradas em todo o sistema, com exceção das parcelas de pagamento de frete, que são tratadas em módulo diferenciado. As parcelas a pagar e receber são cadastradas automaticamente nos módulos a seguir:

- Venda: registra os contratos de venda de cultura e produto; emite relatórios de notas não quitadas, de blocos de algodão não carregados e extrato de saída de cultura com base no módulo "Saída de Produção". Gera parcelas a receber;
- Compra: registra e controla as compras realizadas. As compras podem ter origem através do cadastro de um pedido no módulo "Pedido", que precisará passar por aprovação através de uma cotação de preços pelo módulo "Cotação". Também é possível o cadastro de compras avulsas. Gera parcelas a pagar;
- Despesa: cadastra despesas gerais. Gera parcelas a pagar;
- Empréstimo: registra os empréstimos solicitados ao banco. Gera parcelas a pagar;
- Receitas Diversas: cadastra receitas avulsas para situações que não se encaixam no módulo de venda. Gera parcelas a receber.

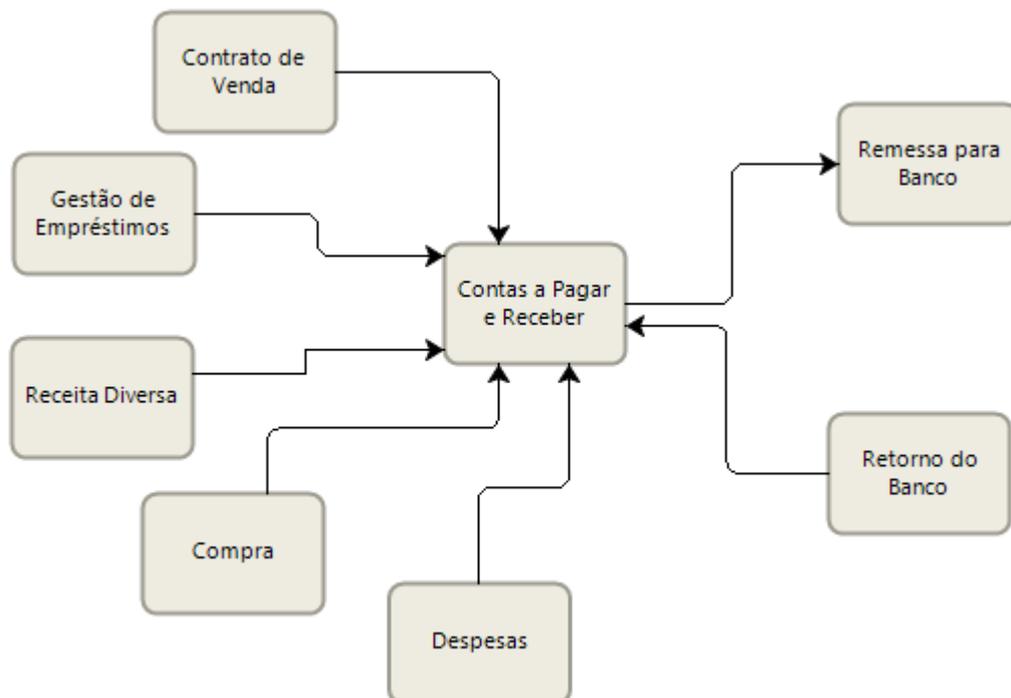


Figura 12: Fluxo do módulo "Contas a Pagar e receber" do sistema GeAgroBit.

O módulo "Contas a Pagar e Receber" deve ser capaz de editar a data de pagamento das parcelas, alterar a data de vencimento das parcelas não quitadas, exibir detalhes das parcelas, registrar pagamentos, mesclar parcelas, estornar pagamentos.

As parcelas podem ter o pagamento registrado pelo próprio módulo "Contas a Pagar e Receber" ou através de arquivos de remessa. Existe o módulo "CNAB Remessa" que gera arquivo de remessa para ser enviado ao banco e o usuário pode carregar o arquivo de retorno do banco no módulo "CNAB Retorno" para confirmar o pagamento em "Contas a Pagar e Receber". Atualmente o uso de arquivos de remessa do sistema GeAgroBit está homologado apenas com o Banco do Brasil e em fase de homologação com o banco Bradesco.

Caso de Uso

Um Caso de Uso descreve as principais funcionalidades do sistema que produz algum resultado, onde há a interação com os usuários em que existe o início e fim em tempo próximo, geralmente executado em minutos (HEUMANN, 2008)

(WAZLAWICK, 2011). Este diagrama é importante para a especificação de requisitos e deve ser escrito de maneira que clientes e desenvolvedores compreendam as funcionalidades de maneira acordada.

A descrição dos casos de uso se encontram disponíveis no ANEXO 4.

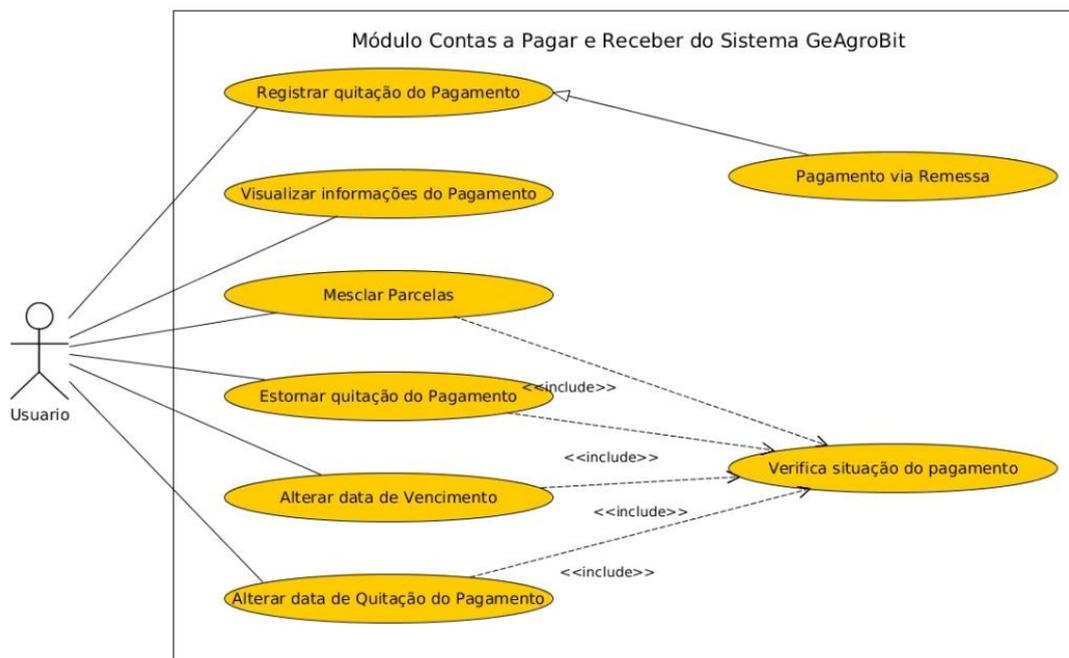


Figura 13: Diagrama de Caso de Uso do módulo "Contas a Pagar e Receber".

Diagrama de Conceito

Um conceito é "a representação da informação complexa que agrega atributos e que não pode ser descrita meramente por tipos alfanuméricos" (WAZLAWICK, 2011). Para entender os conceitos do sistema, tais como "venda", "frete", "compra" e "receita", desenvolveu-se um diagrama de conceitos simples que auxiliou a compreender a relação entre diversos módulos do sistema.



Figura 14: Diagrama de Conceitos do módulo "Contas a Pagar e Receber"

Diagrama Entidade – Relacionamento

O banco de dados do sistema GeAgroBit não possui suas chaves estrangeiras definidas, sendo esse tipo de relação tratada diretamente na aplicação. No ANEXO 2 está disponibilizada a lista de tabelas do banco de dados relacionadas ao módulo.

Teste de desempenho

Foi utilizado para testes de desempenho a classe Monitor do *framework* da InovaBit e o sistema *Neor Profile SQL*. A classe Monitor foi inserida no arquivo `pagamento_frete.ajax.php`, inicio e no final da execução do filtro, conforme ilustrado na figura a seguir.

```

*****
* Iniciando monitor do sistema
*****
include_once($_SESSION['DirBase'] . 'framework/monitor.class.php');
$Monitor = new Monitor();

switch ($_GET['Op'])
{
    case "Fil": ##Filtro

        //Inicia Monitoramento
        try{ $Monitor->inicioMonitoramento(); }catch(Exception $E){}

        //Verificando permisso?
        $Ac->setOpcao($_GET['Op']);
        $Ac->acessoModulo();

        include_once($_SESSION['DirBase'].PACOTE.'/'.MODULO.'/'.MODULO.'.form.php');

        $Form = new PagarReceberForm();
        $PR = new PagarReceber();
        $FPHP = new FuncoesPHP();

        try
        {
            $Form->setEnv("true");
            $Form->setOp($_GET['Op']);
            $Form->getFormFiltro();

            echo($FPHP->formataErro($Form->getErro()).$PR->filtrar($Form));
        }
        catch (Exception $E)
        {
            echo($E->getMessage());
        }
        try{ $Monitor->fimMonitoramento(); }catch(Exception $E){}
        break;
}

```

Figura 15: Uso da classe Monitor no filtro do módulo "Contas a Pagar e Receber" do sistema GeAgroBit.

O sistema Neor Profile SQL monitora o servidor MySQL, e a funcionalidade utilizada foi a de execução de *query* separadamente, exibindo tempo de execução, linhas retornadas e EXPLAIN.

As execuções durante o teste foram realizadas adicionando `SQL_NO_CACHE` às instruções SQL. Este comando não verifica ou armazena cache da consulta, permitindo assim resultados mais realistas do desempenho durante as execuções de teste (MYSQL).

Descrição da Solução e Resultados

As alterações feitas no filtro do módulo "Contas a Pagar e Receber" foram realizadas em duas etapas: Melhoria no código da função que retorna o código SQL a ser executado e melhoria na função que retorna a *subquery* usada na *query* principal.

Melhoria de código

1. Código SQL para legenda

Verificou-se que a mesma *query* SQL era realizada duas vezes com o objetivo de montar a lista de pagamentos e outra para montar a legenda com valores totalizadores. O primeiro código é passado para uma função do *framework* que executará a *query* para montar a lista de pagamentos exibida no módulo. O segundo código é passado para a função `getLegenda` da classe `PagarReceber` que retorna código HTML com os valores totais dos pagamentos filtrados. Constatou-se que as informações solicitadas eram além das necessárias para a função que gera a legenda. Foi realizada a adição de um parâmetro na função que retorna o código SQL a ser executado que realiza verificação se o código será ou não utilizado na busca de dados para legenda (Figura 16). Caso não seja para uso na função `getLegenda` os dados adicionais são inseridos no código e assim serão buscados no banco de dados.

```

public function filtrarSql($ObjForm, $Legenda = NULL) {
    $Fil = new Filtrar($ObjForm);
    $AcAdm = AcessoAdm::inicioAdm();
    $this->operadores();
    $this->ObjForm = $ObjForm;

    if ($this->getFiltroValida()) {...12 linhas}
    $CamposFiltrar = "";
    if (empty($Legenda)) {
        $CamposFiltrar = " , COALESCE(c.CompraCod, xp.CompraCod $Compra) AS CompraCod,
        COALESCE(c.CotacaoCod, xa.CotacaoCod $CotacaoCod) AS CotacaoCod,
        COALESCE(c.PedidoCod, xa.PedidoCod $PedidoCod) AS PedidoCod,
        COALESCE(d.ContratoCod, xe.ContratoCod $Contrato) AS ContratoCod,
        f.UnidadeNegocioNome, a.PagarReceberCod AS NotaFiscalCod,
        IF(e.LancamentoContabilCod IS NOT NULL,
        CONCAT('Despesa: ',b.LancamentoContabilTitulo, ' - ', a.PagarReceberParcela),
        CONCAT(b.LancamentoContabilTitulo, ' - ', a.PagarReceberParcela) ) AS Documento,
        CONCAT(COALESCE(g.FornecedorNome, h.ClienteNome), '<br>',
        COALESCE( CONCAT( ' ', COALESCE(kd.Nome, ke.RazaoSocial), ' ' ), '&nbsp;' ) ) AS FornecedorCliente,
        a.PagarReceberDataVencimento, a.PagarReceberDataPagamento, a.PagarReceberRemessa,
        IF ((em.EmprestimoCod IS NOT NULL), CONCAT('EM', em.EmprestimoCod), '-') AS EmprestimoCod,
        IF ((rd.ReceitaDiversaCod IS NOT NULL), CONCAT('RD', rd.ReceitaDiversaCod ), '-') AS ReceitaDiversaCod,
        (a.PagarReceberCod) AS PagarReceberCodMesclado,
        COALESCE( e.CompraDespesaCod $CompraDespesaCod ) AS CompraDespesaCod,
        COALESCE(vp.FretePagamentoCod $FretePagamentoCod) AS FretePagamentoCod,
        b.LancamentoContabilData, COALESCE(c.CompraDataFechamento, e.CompraDespesaDataEmissao,
        em.EmprestimoDataEmissao, d.ContratoData, rd.ReceitaDiversaData) AS DataEmissao,
        IF(a.PagarReceberValorDocumentoTipo = 'R'), 'Real', 'Dolar') AS MoedaRegistro,
        CASE a.PagarReceberTipoLancamento
        WHEN 'P' THEN IF((a.PagarReceberRemessa != 'N'),
        IF((a.PagarReceberSituacaoPago = 'S'), 'Pago - Para Remessa', 'A Pagar - Para Remessa'),
        IF((a.PagarReceberSituacaoPago = 'S'), 'Pago', 'A Pagar')) )
        WHEN 'R' THEN IF((a.PagarReceberSituacaoPago = 'S'), 'Recebido', 'A Receber'))
        END AS PagarReceberSituacaoPagoGrid";

    $Sql = "SELECT $DISTINCT a.PagarReceberCod, a.PagarReceberTipoLancamento, a.PagarReceberSituacaoPago, a.PagarReceberValorDocumento,
    a.PagarReceberValorPago, a.PagarReceberValorDocumentoTipo $CamposFiltrar";
}

```

Figura 16: Função que gera código SQL após modificação de atributo para verificação de uso na função `getLegenda` a fim de diminuir a quantidade de dados retornados, conforme necessidade de uso.

2. Limitando a *subquery*

A *query* SQL principal da função `filtrar` possui uma *subquery* que realiza busca de dados de parcelas mescladas. Esta *subquery* é apenas adicionada a *query* principal, através da função `getDadoOrigemMesclagemSql` quando as opções "Código da compra", "Código da cotação", "Código do pedido", "Código da venda", "Código da despesa" e/ou "Código do empréstimo" no filtro são

preenchidos, sendo esta condição verificada na função "getFiltroValida". Contudo realizando o comando EXPLAIN na *query* principal foi verificada que a *subquery* retornava um número excessivo de registros a cada busca realizada.

Foi identificado que a quantidade de registros retornada pela *subquery* era causada pela busca de informações sem condições na cláusula WHERE, fazendo assim uma busca completa nas tabelas relacionadas retornando pouco mais de 40 mil registros, em todas as linhas buscadas na *query* principal.

Analisando o código do módulo pode-se verificar que a função "getDadoOrigemMesclagemSql" é utilizada em outros momentos do sistema. Desta forma foi realizada a adição de um parâmetro de padrão nulo na função, passando a variável "\$ObjForm", que contém dados do formulário de filtro, e inseridos usos das funções do *framework* que adicionam texto em forma de cláusula WHERE utilizando dados do filtro, conforme ilustrado na Figura 17.

```

561 public function getDadoOrigemMesclagemSql($PagarReceberCod = NULL, $ObjForm = NULL) {
562     $Fil = new Filtrar($ObjForm);
563     //Código SQL SubQuery
564     /*...18 linhas */
565     if (!empty($PagarReceberCod)) {
566         $Sql .= " AND a.PagarReceberCod = $PagarReceberCod";
567     }
568     if (!empty($ObjForm)) {
569         $ObjForm->setCampoRetorna("CompraChave", implode(preg_split('/[^:digit:]/', $ObjForm->getCampoRetorna("CompraChave"))));
570         $Sql .= $Fil->getStringSql("CompraChave", "COALESCE(c.CompraCod, xp.CompraCod)", "Inteiro");
571         $ObjForm->setCampoRetorna("ContratoChave", implode(preg_split('/[^:digit:]/', $ObjForm->getCampoRetorna("ContratoChave"))));
572         $Sql .= $Fil->getStringSql("ContratoChave", "COALESCE(d.ContratoCod, xe.ContratoCod)", "Inteiro");
573         $ObjForm->setCampoRetorna("CompraDespesaChave", implode(preg_split('/[^:digit:]/', $ObjForm->getCampoRetorna("CompraDespesaChave"))));
574         $Sql .= $Fil->getStringSql("CompraDespesaChave", "e.CompraDespesaCod", "Inteiro");
575         $ObjForm->setCampoRetorna("FreteChave", implode(preg_split('/[^:digit:]/', $ObjForm->getCampoRetorna("FreteChave"))));
576         $Sql .= $Fil->getStringSql("FreteChave", "xp.FretePagamentoCod", "Inteiro");
577         $ObjForm->setCampoRetorna("PedidoChave", implode(preg_split('/[^:digit:]/', $ObjForm->getCampoRetorna("PedidoChave"))));
578         $Sql .= $Fil->getStringSql("PedidoChave", "COALESCE(c.PedidoCod, xa.PedidoCod)", "Inteiro");
579         $ObjForm->setCampoRetorna("CotacaoChave", implode(preg_split('/[^:digit:]/', $ObjForm->getCampoRetorna("CotacaoChave"))));
580         $Sql .= $Fil->getStringSql("CotacaoChave", "COALESCE(c.CotacaoCod, xa.CotacaoCod)", "Inteiro");
581     }
582     return $Sql;
583 }

```

Figura 17: Alteração na função " getDadoOrigemMesclagemSql " para limitar a quantidade de registros retornadas, otimizando a *query* principal da função "FiltrarSql".

Esta alteração resultou em melhorias explícitas no desempenho do uso do filtro. A quantidade de registros retornados pela *subquery* diminuiu de mais de cinquenta mil registros para apenas um. Utilizando a classe Monitor antes de realizar alterações em "getDadoOrigemMesclagemSql" o tempo de resposta ao se realizar uma busca por código de compra era de 2495,5 segundos (aproximadamente 41 minutos). Após as alterações realizadas o tempo diminuiu para 3,5744 segundos, o que significa um ganho de desempenho de 99%. Esta melhoria é significativa para o

usuário, que comumente precisa realizar busca das parcelas a pagar e receber pelo número identificador das compras, vendas, empréstimos, despesas cotação e pedido.

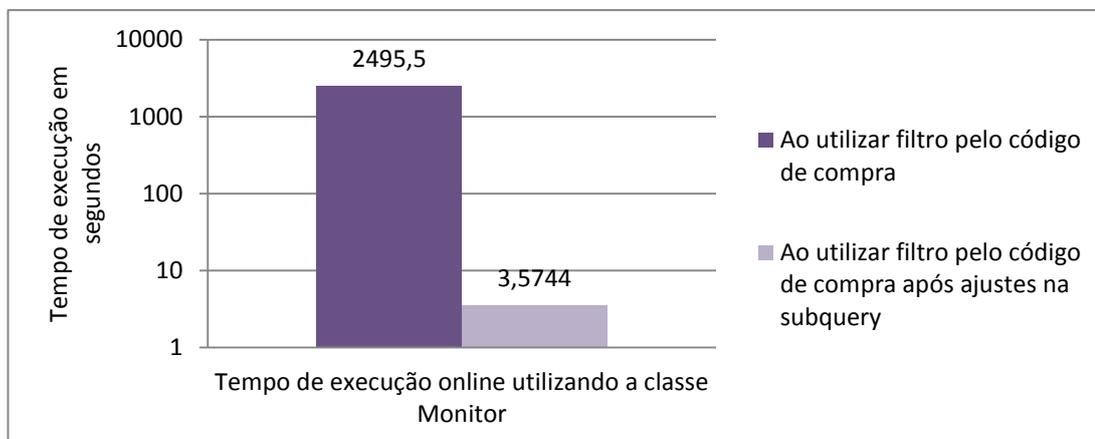


Figura 18: Gráfico ilustrando a melhoria na performance do módulo "Contas a Pagar e Receber após ajustes na *subquery*.

3.3.2. Módulo "Pagamento de Frete"

O módulo "Pagamento de Frete" cadastra a quitação de parcelas a pagar referentes a serviços de frete utilizados, bem como as informações de confirmação de peso entregue para cálculo de diferenças no valor a ser pago. As parcelas de frete a pagar são cadastradas automaticamente em formulários onde o peso é informado através da utilização de balança nos módulos a seguir:

- Entrada de Produto: registra entradas de produto provenientes do módulo "Compra" ou "Transferência de produto".
- Cultura em Trânsito: Registra as saídas de cultura diretamente da lavoura, sem dar entrada no estoque da unidade de negócio produtora. Esta produção será encaminhada para outra unidade de negócio, que fará o cadastro da entrada da produção em seu estoque.
- Saída de Produção: Registra as saídas de cultura para o cliente.
- Utilização de Cultura: O módulo de utilização de cultura permite o registro de uso da cultura pela própria empresa. Existem as opções:
 - Consumo próprio: a fazenda faz o consumo da cultura;

- Transferência de unidade de Produção: registra a troca da unidade de produção em que a cultura foi produzida;
- Transferência de armazém: é registrada a troca de armazém (sendo estes em outras unidades de negócio ou não);
- Transferência de Unidade de Negócio: registra a troca da responsabilidade da cultura para outra unidade de Negócio, permitindo assim que esta nova unidade de negócio a comercialize ou cadastre utilizações para ela.

Ao abrir o módulo "Pagamento de Frete" é exibida uma tela com as informações relevantes para identificação dos pagamentos: código identificador de compra, código identificador de pedido, código identificador de venda, código identificador de transferência de cultura, código identificador de transferência de produto, número de nota fiscal, número de ticket, placa, nome da empresa transportadora, nome do motorista, quantidade do produto/cultura de entregue ao transportador na origem, quantidade do produto/cultura entregue ao transportador no destino, diferença entre a quantidade entregue na origem e no destino, nome do produto, situação do pagamento, origem do registro no sistema, data de recebimento da cultura/produto, data pagamento do frete, unidade de negócio de origem.

Caso de Uso

Os casos de uso do módulo "Pagamento de Frete" são simples, uma vez que o módulo dispõe de poucas funcionalidades. A descrição dos casos de uso se encontram disponíveis no ANEXO 5.

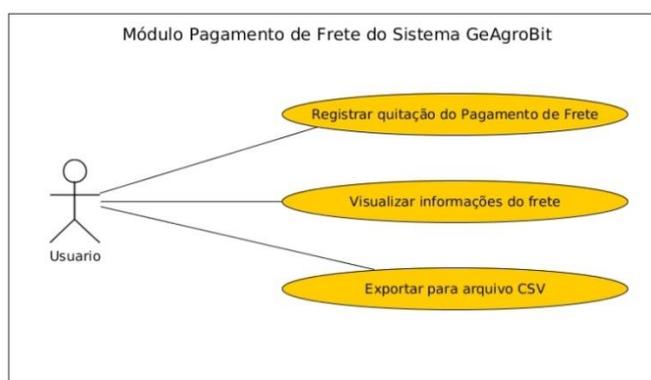


Figura 19: Diagrama de Caso de Uso do módulo "Pagamento de Frete"

Diagrama de Conceito

Para entender os conceitos do módulo "Pagamento de Frete", tais como "parcela_frete" desenvolveu-se um diagrama de conceitos simples que auxiliou a compreender a relação entre diversos módulos do sistema.

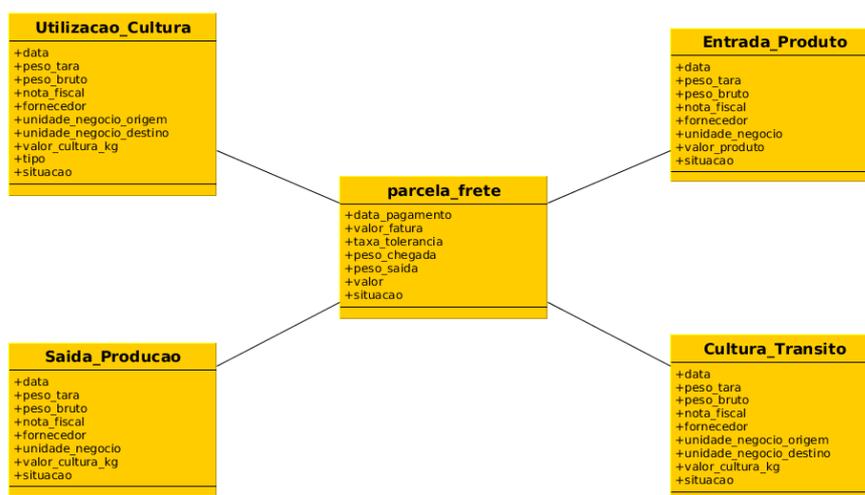


Figura 20: Diagrama de Conceitos do módulo "Contas a Pagar e Receber"

Diagrama Entidade – Relacionamento

O banco de dados do sistema GeAgroBit não possui suas chaves estrangeiras definidas, sendo esse tipo de relação tratada diretamente na aplicação. No ANEXO 3 está disponibilizada a lista de tabelas do banco de dados relacionadas ao módulo.

Teste de Desempenho

Os testes de desempenho foram realizados utilizando o *software* Neor Profile SQL e a classe Monitor do *framework* InovaBit, inserida no arquivo pagar_receber.ajax.php, conforme ilustrado abaixo:

```

24  /*****
25  * Iniciando monitor do sistema
26  *****/
27  include_once($SESSION['DirBase'] . 'framework/monitor.class.php');
28  $Monitor = new Monitor();
29
30  switch ($_GET['Op'])
31  {
32      case "Fil*": ##Filtro
33          //Inicia Monitoramento
34          try{ $Monitor->inicioMonitoramento(); }catch(Exception $E){}
35
36          //Verificando permissões
37          $Ac->setOpcao($_GET['Op']);
38          $Ac->acessoModulo();
39
40          include_once($SESSION['DirBase'].PACOTE.'/'.MODULO.'/'.MODULO.'.form.php');
41
42          $Form = new FRPagamentosForm();
43          $FRPag = new FRPagamentos();
44          $FPHP = new FuncoesPHP();
45
46          try
47          {
48              $Form->setEnv("true");
49              $Form->setOp($_GET['Op']);
50              $Form->getFormFiltro();
51
52              echo($FPHP->formataErro($Form->getErro()).$FRPag->filtrar($Form));
53          }
54          catch (Exception $E)
55          {
56              echo($E->getMessage());
57          }
58          //Final Monitoramento
59          try{ $Monitor->fimMonitoramento(); }catch(Exception $E){}
60          break;
61

```

Figura 21: Uso da classe Monitor no filtro do módulo "Pagamento de Frete" do sistema GeAgroBit.

A primeira execução do sistema sem realizar alterações no código além da adição da classe Monitor resultou em uma demora de 162,19 segundos para abrir a página do módulo. A utilização dos filtros também resultou em demora excessiva, sem nenhuma diferença como o ocorrido no módulo "Contas a Pagar e Receber". Os resultados encontram-se disponibilizados no Quadro 2.

Quadro 2: Resultado do monitoramento do filtro em "Pagamento de Frete" do sistema GeAgroBit no servidor local para verificar o problema informado.

Ação	Tempo de execução em segundos utilizando classe Monitor	Tempo de execução em segundos usando Neor Profile SQL
Abrir Módulo	162,19	101,19
Filtrar pelo código de compra	159,90	100,10
Filtrar por unidade de Negócio	160,72	101,01
Filtrar por Produto	161,66	103,10
Filtrar por Cultura	162,56	100,01
Filtrar por Nota Fiscal	160,34	101,00
Filtrar por Produto e Unidade de Negócio	161,40	102,01
Filtrar usando todos os campos (exceto cultura)	161,44	101,06

Descrição da Solução e Resultados

A primeira medida tomada foi verificar a *engine* utilizada pelas tabelas do banco de dados envolvidas na função que buscam os dados iniciais do módulo. Constatou-se que as tabelas `entrada_cultura_utilizacao` e `entrada_cultura_utilizacao_balanca` eram `myISAM`, ou seja, não suportavam transações e a cada ação de atualização, inserção ou deleção havia bloqueio a nível de tabela, o que pode causar problemas de concorrência, tornando as ações e busca de dados mais lentas. Desta forma, atualizou-se a *engine* das tabelas referidas para `InnoDB`, padrão utilizado pelo sistema `GeAgroBit`, que bloqueia a nível de linha e permite transações.

A *query* utilizada para trazer as informações era composta de uma busca dentro de uma *subquery* contendo esta quatro uniões de tabela utilizando `UNION` para que todos os dados necessários pudessem ser retornados (Figura 22). Todas estas uniões são realizadas dentro de uma *subquery* para possibilitar o uso mais simples das funções do *framework* (Figura 23) para utilização dos campos do filtro na cláusula `WHERE`, além de código mais limpo e fácil de manutenção. Contudo esta solução traz como malefício a busca completa de todos os dados para depois o resultado ser filtrado pela cláusula `WHERE`. Desta forma mesmo que índices fossem adicionados o código SQL estaria solicitando a busca de todos os dados para depois selecioná-los do resultado desta *subquery*.

```
388 SELECT * FROM (
389     SELECT campos FROM saida_producao
390     UNION
391     SELECT campos FROM entrada_produto
392     UNION
393     SELECT campos FROM transferencia_cultura
394     UNION
395     SELECT campos FROM cultura_utilizacao
396 ) AS x WHERE x.campos = 241;
```

Figura 22: Exemplo da estrutura da consulta SQL da função filtrar em "Pagamento de Frete".

```

180 $Sql .= $Fil->getStringSql("Fornecedor", "x.Transportador", "Texto");
181
182 $ObjForm->setCampoRetorna("Compra", implode(preg_split('/[^[:digit:]]/', $ObjForm->getCampoRetorna("Compra"))));
183 $Sql .= $Fil->getStringSql("Compra", "x.CompraCod", "Inteiro");
184
185 $ObjForm->setCampoRetorna("Pedido", implode(preg_split('/[^[:digit:]]/', $ObjForm->getCampoRetorna("Pedido"))));
186
187 $Sql .= $Fil->getStringSql("Pedido", "x.PedidoCod", "Inteiro");
188 $Sql .= $Fil->getStringSql("Produto", "x.CulturaProduto", "Texto");
189
190 $ObjForm->setCampoRetorna("Venda", implode(preg_split('/[^[:digit:]]/', $ObjForm->getCampoRetorna("Venda"))));
191 $Sql .= $Fil->getStringSql("Venda", "x.VendaCod", "Inteiro");
192
193 $Sql .= $Fil->getStringSql("Cultura", "x.CulturaProduto", "Texto");
194 $Sql .= $Fil->getStringSql("FretePago", "x.SituacaoFiltro", "Texto");
195 $Sql .= $Fil->getStringSql("NotaFiscal", "x.NotaFiscal", "Texto");
196 $Sql .= $Fil->getStringSql("DataRecebimento", "DATE_FORMAT(x.DataRecebimento, '%Y/%m/%d')", "Data");
197 $Sql .= $Fil->getStringSql("DataPagamento", "DATE_FORMAT(x.DataPagamento, '%Y/%m/%d')", "Data");
198 $Sql .= $Fil->getStringSql("OrigemLancamento", "x.Origem", "Texto");
199 $Sql .= $Fil->getStringSql("Ticket", "x.TICKET", "Texto");
200 $Sql .= $Fil->getStringSql("UnidadeNegocio", "x.UNNome", "Texto");

```

Figura 23: Utilização da função `getStringSql` do *framework* InovaBit para adição de condições nas cláusulas `WHERE`.

O primeiro passo para otimização desta consulta foi reescrever o código SQL que filtra os dados. As uniões foram trocadas por junções, e os campos selecionados foram verificados utilizando `COALESCE`, uma função SQL que retorna o primeiro valor não nulo da lista de campos passados como parâmetro, ou nulo quando todos forem vazios. Contudo, devido a grande quantidade de junções, a consulta permaneceu com tempo de resposta insatisfatório e maior do que a consulta anterior, levando em testes locais cerca de 656 segundos, ou seja, 408,62% a mais de tempo de execução em ambiente de teste utilizando a classe `Monitor`.

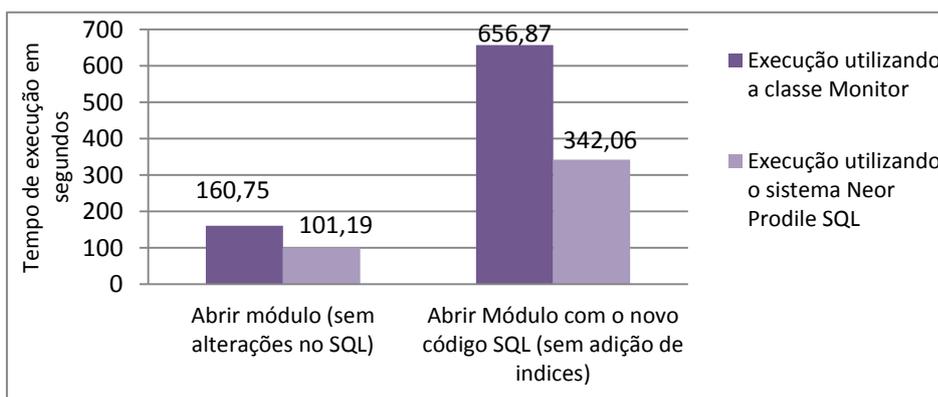


Figura 24: Comparação do desempenho das consultas SQL da função `filtrar` do módulo "Pagamento de Frete" do sistema `GeAgroBit` em ambiente de teste.

Realizando o comando `EXPLAIN` no novo código SQL no ambiente de teste pode-se constatar que as tabelas `tiket_controle`, `frete_pagamento_registro`, `entrada_cultura_utilizacao` e a *subquery* para busca do número de nota fiscal das utilizações de cultura percorrem toda a tabela para a busca de uma informação, pois o

tipo de busca é *ALL*. Desta forma fez-se necessária a adição de índices nos campos relacionados às junções, presentes nas cláusulas *ON*.

Analisando o resultado do comando EXPLAIN e os índices da tabela frete_pagamento_registro, frete_pagamento_registro, demonstrados

Quadro 3, verificou-se a necessidade da criação de índices nas colunas relacionadas às junções com as tabelas de transferência de cultura e utilização de cultura. Foi criado índice do tipo *INDEX* na coluna "TransferenciaCulturaCod" e outro "EstoqueCulturaUtilizacaoCod". Com essa modificação o tipo de busca na tabela frete_pagamento_registro em junção com estoque_cultura_utilizacao mudou para *ref*, tipo de acesso que utiliza índices para achar a informação mais rapidamente, e diminuiu de 12.839 linhas examinadas para 45. Já a linha do resultado do *EXPLAIN* relacionada à junção com a tabela transferencia_cultura diminuiu de 12.839 para apenas uma linha examinada e o tipo de acesso mudou também para *ref*.

Quadro 3: Índices da tabela frete_pagamento_registro

CHAVES				
Nome da chave	Tipo	Único	Coluna	Quantidade
PRIMARY	BTREE	Sim	FretePagamentoRegistroCod	14501
FretePagamentoCod	BTREE	Não	FretePagamentoCod	3625
EntradaProdutoLogCod	BTREE	Não	EntradaProdutoLogCod	14501
SaidaProducaoCod	BTREE	Não	SaidaProducaoCod	2
CompraCod	BTREE	Não	CompraCod	172
ContratoCod	BTREE	Não	ContratoCod	2

A tabela entrada_cultura_utilizacao possuía apenas a sua chave primária como índice, e no comando *EXPLAIN* pode-se verificar que a consulta percorre toda a sua tabela, num total de 493 linhas. Foi realizada a adição do índice de múltiplas colunas usando as colunas "EstoqueCulturaUtilizacaoCod" e "EntradaCulturaUtilizacaoStatus". Desta forma a consulta passou a buscar apenas uma linha, e o tipo foi alterado para *ref*.

Foi realizada a criação de índice na tabela "ticket_controle" utilizando a coluna "TicketOrigem", o que mudou o tipo de consulta de *ALL* para *range*, que não faz uma busca completa na tabela, inicia em algum ponto e retorna as linhas que satisfazem o intervalo desejado (SCHWARTZ, ZAITSEV e TKACHENKO, 2012). Assim, a quantidade de linhas percorridas diminuiu de 62.511 para 33.998.

Apesar de ser possível editar notas fiscais no sistema, as tabelas que armazenam estes dados não recebem nenhuma operação de atualização (UPDATE) ou remoção (DELETE). Desta forma fez-se necessário a continuidade da *subquery* para a busca de informações da nota fiscal. Este é um problema que deverá ser sanado futuramente, visando a melhoria em diversos outros módulos do sistema.

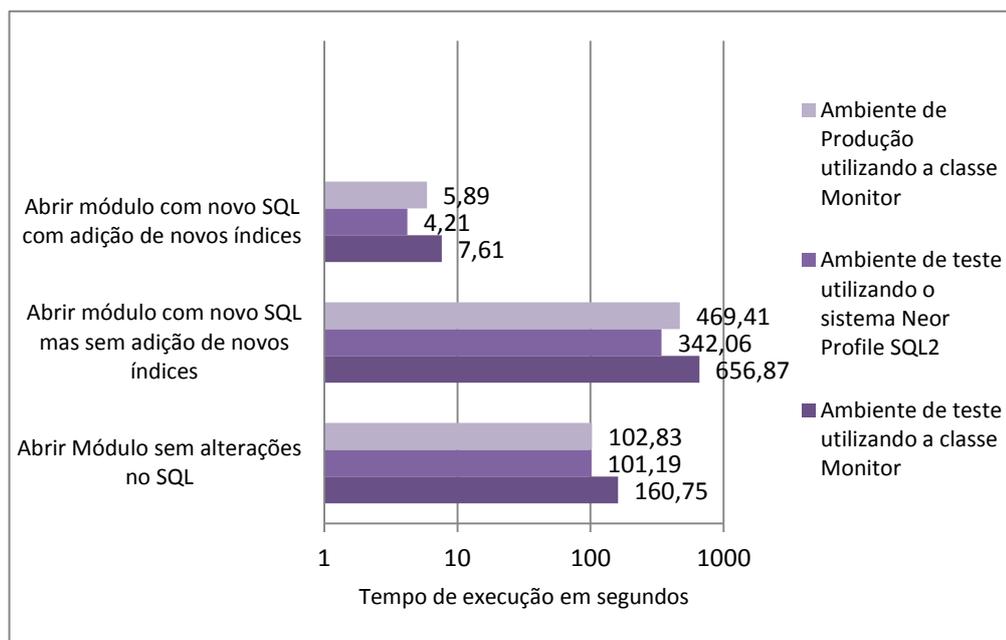


Figura 25: Comparação do tempo de execução da consulta da função filtrar no módulo "Pagamento de Frete".

Todas as mudanças resultaram em 92% de melhora em tempo de execução para carregamento da página. Apesar do servidor de produção não aceitar as conexões através do Neor Profile SQL para realização de medição, estas foram feitas utilizando a classe Monitor, e mostra claramente o aumento da velocidade ao utilizar a função filtrar no módulo "Pagamento de Frete", conforme Figura 25.

Estas melhorias afetaram diretamente a rotina dos usuários, que instantaneamente notaram diferenças, confirmando o êxito do trabalho.

4. DIFICULDADES ENCONTRADAS

Durante o estágio supervisionado uma das dificuldades iniciais foi a falta de conhecimento específico sobre índices em banco de dados. Apesar dos conhecimentos básicos sobre banco de dados adquiridos durante o curso de graduação, fez-se necessário um estudo mais direcionado ao funcionamento de índices, e posteriormente ao uso de índices em SGBD MySQL.

Outro problema encontrado foi a falta de documentação completa do *framework* da InovaBit, o que tornou o entendimento dos códigos do sistema GeAgroBit mais demorado.

Também devido a falta de documentação apropriada a compreensão do fluxo de processos dos módulos abrangidos tornou mais demorada. O escopo do trabalho foi ajustado para que atualizações necessárias na documentação fossem realizadas, o que resultou em descrição de requisitos, casos de uso e diagramas de conceito que não estavam planejados inicialmente.

Além disso, houve falhas no projeto inicial do sistema em relação a banco de dados, que não se encontra modelado de maneira ideal, contendo campos desnecessários ou mal elaborados, o que acarretou em certa dificuldade de compreender onde os dados encontram-se de fato armazenados.

Por fim, a não atualização do servidor para a versão mais atual do banco de dados MySQL ocorreu pois a InovaBit não administra os servidores de produção, cabendo o escopo de atualização a outra empresa. Esta empresa, já no final do projeto, disponibilizou o ambiente para que fosse realizada a atualização, contudo devido a falta de conhecimentos técnicos para atualização de servidores Debian via terminal remoto, a equipe InovaBit preferiu-se adiar a atualização para que sua realização fosse realizada com maior planejamento.

5. CONSIDERAÇÕES FINAIS

O desenvolvimento deste trabalho visava a documentação de maneira sucinta dois módulos do sistema GeAgroBit como forma de apoio na compreensão do sistema e desenvolver uma solução para o problema referente a performance na manipulação de dados.

Pretendia-se ainda que através da elaboração deste trabalho o aluno em estágio aprendesse os conceitos relacionados à engenharia de *software* e banco de dados, bem como aprender a trabalhar na linguagem de programação PHP e com as ferramentas NetBeans, banco de dados MySQL e PhpMyAdmin, o que ocorreu durante o desenvolvimento deste trabalho.

Com o estudo dos conceitos descritos no primeiro capítulo e a utilização das ferramentas listadas acima a atualização da documentação e o desenvolvimento da solução dos problemas de performance nos módulos "Contas a Pagar e Receber" e "Pagamento de Frete" foram realizados, permitindo aos clientes da InovaBit o aumento de satisfação durante a utilização do sistema, e ao aluno em estágio a aprendizagem e experiência, desta forma atingindo os objetivos listados na introdução deste documento.

Sabe-se que muitas outras ocorrências acontecem no sistema GeAgroBit e este trabalho evidencia um caminho a ser tomado em futuras otimizações de outros módulos. Além disso, a utilização da classe Monitor do *framework* mostrou-se uma ferramenta capaz de auxiliar na medição de desempenho do sistema em geral, podendo ser melhorada e adicionada em rotinas que auxiliem a equipe InovaBit a identificar situações onde o sistema apresente desempenho insatisfatório, possibilitando melhorias e trabalhos futuros relacionados.

REFERÊNCIAS BIBLIOGRÁFICAS

ACTIAN. Products Versant. **Actian**. Disponível em: <<http://www.actian.com/products/operational-databases/versant/>>. Acesso em: 22 out. 2014.

BOSCARIOLI, C. et al. Uma reflexão sobre Banco de Dados Orientados a Objetos. **IV CONGED**, 2006. Disponível em: <http://conged.deinfo.uepg.br/artigo4.pdf>.

BOURQUE, P.; FAIRLEY, R. E. **Guide to the Software Engineering Body of Knowledge. Version 3.0**. : IEEE Computer Society, 2014. www.swebok.org.

CURIOSO, A.; BRADFORD, R.; GALBRAITH, P. **Expert PHP and MySQL**. Indianapolis: Wiley Publishing, 2010.

DATE, C. J. **Introdução a sistemas de bancos de dados**. 8º. ed. Rio de Janeiro: Campus, 2004.

DINIZ, M. S. **Otimização e desempenho de consultas no banco de dados MySQL**. Universidade Federal de Mato Grosso. Cuiabá. 2011. Monografia apresentada ao Curso de Pós-Graduação Lato Sensu, Especialização em Banco de Dados.

FERREIRA, T. Otimização de Consultas MySQL - Parte I. **iMasters**, 15 out. 2009. Disponível em: <http://imasters.com.br/artigo/14624/MySQL/otimizacao_de_consultas_MySQL_parte_01/>. Acesso em: 24 out. 2014.

GOMES, R. S. R. **Processamento de Consultas em Sistema de Gerenciamento de Banco de Dados**. Universidade Federal de Mato Grosso. Cuiabá. 2009. Monografia apresentada ao Curso de Ciência da Computação.

HEUMANN, J. **Tips for writing good use cases**. Somers: IBM Rational *Software*, 2008.

HEUSER, C. A. **Projeto de Banco de Dados**. 4º. ed. Porto Alegre: Sagra, 1998. Instituto de Informática da UFRGS.

INTERSYSTEMS. Banco de dados mais rápido do mundo. **InterSystems Brasil**. Disponível em: <<http://www.intersystems.com.br/produtos/cache>>. Acesso em: 22 out. 2014.

MACÁRIO, C. G. N.; BALDO, S. M. O Modelo Relacional, 2005. Disponível em: <<http://www.ic.unicamp.br/~geovane/mo410-091/Ch03-RM-Resumo.pdf>>. Acesso em: 22 20 2014.

MYSQL. Manual de referência MySQL. **MySQL**. Disponível em: <<http://dev.MySQL.com/doc/refman/5.0/en/index.html>>. Acesso em: 22 out. 2014.

NAVATHE, S.; ELMASRI, R. **Sistemas de Banco de Dados**. 6°. ed. São Paulo: Pearson Education, 2011.

OBJECT MANAGEMENT GROUP. What is UML? **OMG UML**, 10 set. 2014. Disponível em: <http://www.omg.org/gettingstarted/what_is_uml.htm>. Acesso em: 11 out. 2014.

PHP GROUP. Manual do PHP. **Microtime**. Disponível em: <http://php.net/manual/pt_BR/function.microtime.php>. Acesso em: 22 nov. 2014.

POSTGRESQL. Introdução e Histórico. **PostgreSQL Wiki**. Disponível em: <https://wiki.postgresql.org/wiki/Introdução_e_Histórico>. Acesso em: 22 out. 2014.

ROB, P.; CORONEL, C. **Sistemas de Banco de Dados: Projeto, Implementação e Administração**. 8°. ed. Cengage: CENGAGE Learning, 2011.

SCHWARTZ, B.; ZAITSEV, P.; TKACHENKO, V. **High Performance MySQL**. 3°. ed. Sebastopol: O'Reilly Media, 2012.

SOMMERVILLE, I. **Engenharia de Software**. 8°. ed. São Paulo: Persson, 2007. Tradução Selma Shin Melnikoff; Reginaldo Arakaki; Edilson de Andrade Barbosa.

TAKAI, O. K.; ITALIANO, I. C.; FERREIRA, J. E. **Introdução a Banco de Dados**. São Paulo: DCC-IME-USP, 2005.

WAZLAWICK, R. S. **Análise e projeto de Sistemas de Informação Orientados a Objetos**. 2ª. ed. Rio de Janeiro: Elsevier, 2011.

ANEXOS

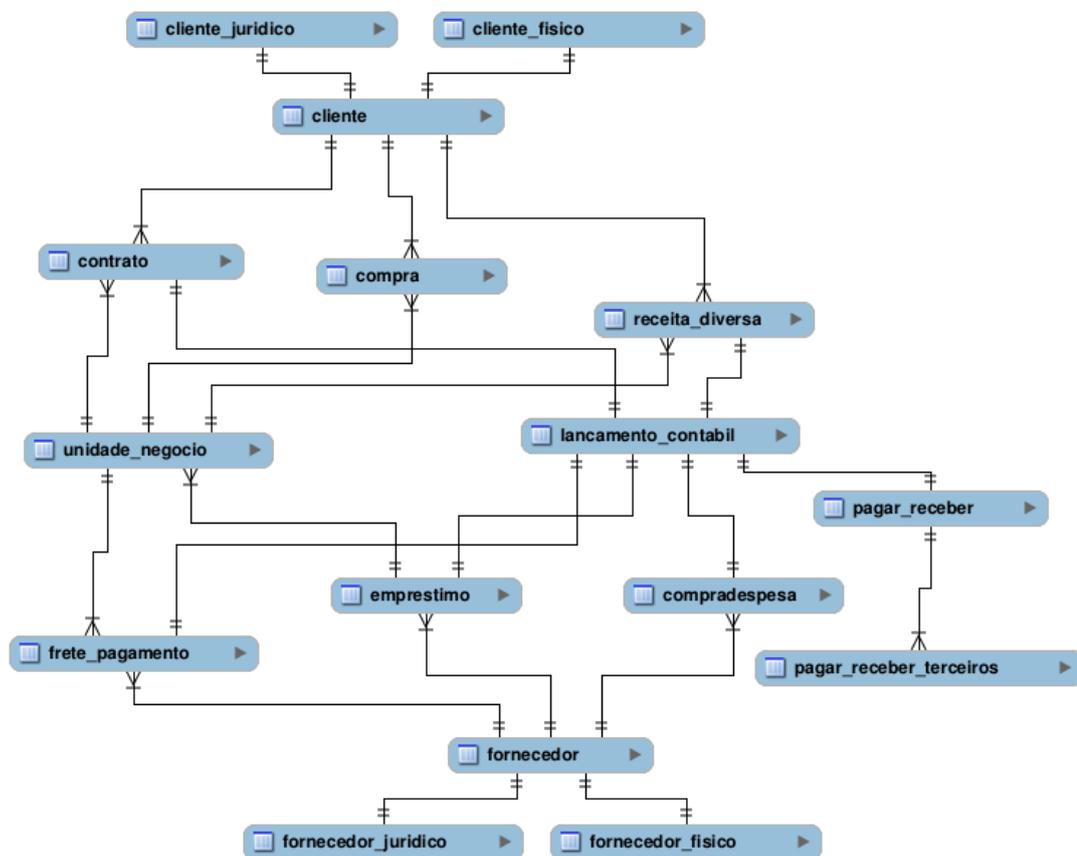
ANEXO 1: Classe Monitor

Código da classe Monitor do *framework* InovaBit, utilizada para realizar medição de desempenho de código.

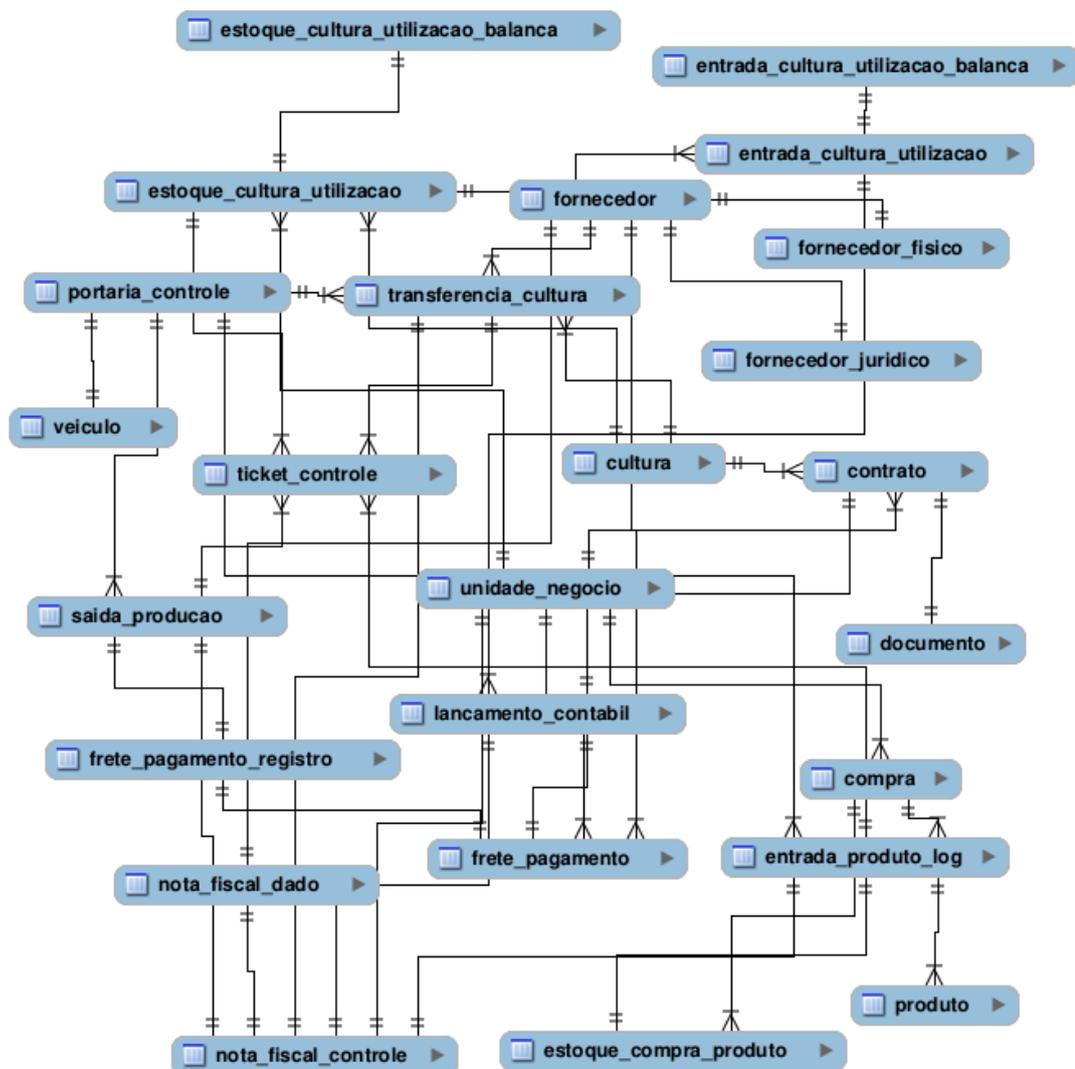
```

1 <?php
2 class Monitor
3 {
4     private $TempoInicial, $TempoFinal;
5
6     private $MonitorCod;
7
8     public function inicioMonitoramento()
9     {
10         //Inicia Conexão
11         $Con = Conexao::conectar();
12
13         //Definindo Código do Módulo
14         $ModuloCod = $Con->execRLinha("SELECT ModuloCod FROM _modulos WHERE ModuloNome = '".MODULO.'");
15
16         //Ação
17         $MonitorAcao = $_GET['Op'];
18
19         //Env
20         $MonitorEnv = $_GET['Env']? 'true' : 'false';
21
22         //SisReg
23         if(is_array($_REQUEST['SisReg']))
24             $MonitorRegistroCod = var_export(array_keys($_REQUEST['SisReg']), true);
25
26         $Sql = "INSERT INTO _monitor (UsuarioCod, ModuloCod, MonitorAcao, MonitorTempo, MonitorData, MonitorRegistroCod, MonitorEnv)
27             VALUES ('.$_SESSION['UsuarioCod'].', ".$ModuloCod."', ".$MonitorAcao.", NULL, now(), ".$MonitorRegistroCod."', ".$MonitorEnv.')";
28
29         $Con->executar($Sql);
30
31         //Código Gerado
32         $this->MonitorCod = $Con->ultimoInsertId();
33
34         list($Usec, $Sec) = explode(" ", microtime());
35
36         $this->TempoInicial = ((float)$Usec + (float)$Sec);
37     }
38
39     public function fimMonitoramento()
40     {
41         list($Usec, $Sec) = explode(" ", microtime());
42
43         $this->TempoFinal = ((float)$Usec + (float)$Sec);
44
45         $this->salvaMonitoramento();
46     }
47
48     public function salvaMonitoramento()
49     {
50         //Inicia Conexão
51         $Con = Conexao::conectar();
52
53         //Tempo de Monitoramento
54         $MonitorTempo = substr(($this->TempoFinal - $this->TempoInicial),0,6);
55
56         $Sql = "UPDATE _monitor SET MonitorTempo = ".$MonitorTempo." WHERE MonitorCod = ".$this->MonitorCod;
57
58         $Con->executar($Sql);
59     }
60 }

```

ANEXO 2: Lista de tabelas relacionadas ao módulo "Contas a Pagar e Receber"

ANEXO 3: Lista de tabelas relacionadas ao módulo "Pagamento de Frete"



ANEXO 4: Casos de uso do módulo "Contas a Pagar e Receber"

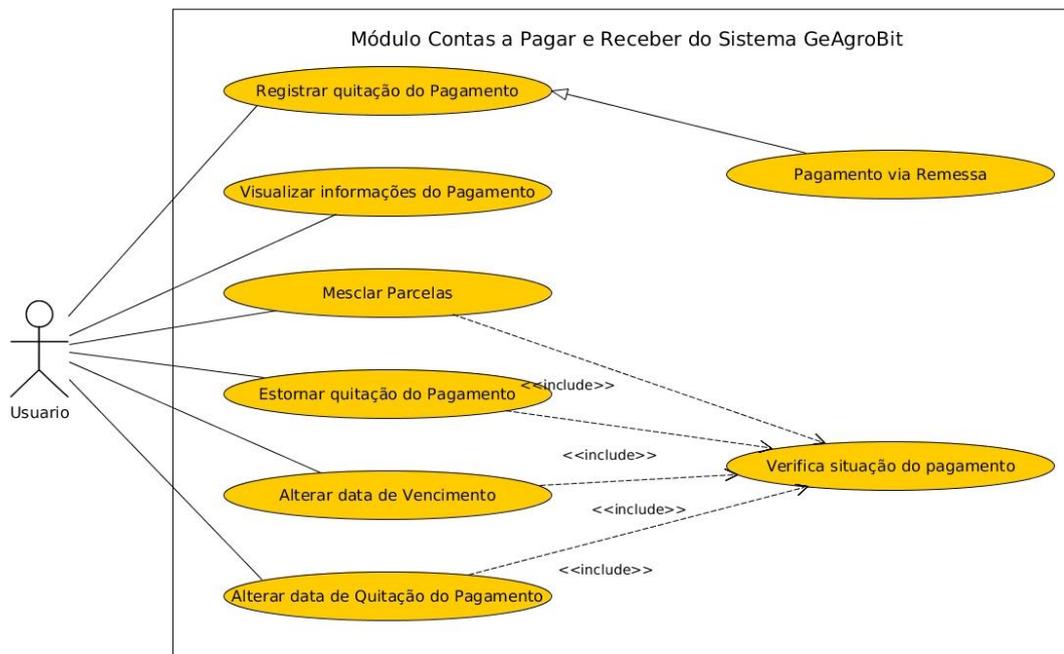


Diagrama de Caso de Uso do módulo "Contas a Pagar e Receber".

- **Caso de Uso: Registrar Quitação de Pagamento**

Ator: Usuário.

Pré-Condições: O Usuário está autenticado no **Sistema** e possui permissão de acesso a opção de módulo **Baixar Pagamento**.

Pós-Condições: É registrada a quitação da parcela (ou parcelas mescladas) selecionada ou a parcela é disponibilizada para quitação nos módulos de Remessa e Retorno.

Fluxo Principal:

- O Usuário seleciona o registro da parcela (ou parcelas mescladas) não quitada desejada.
- O Usuário aciona o comando "Baixar Pagamento".
- O **Sistema** exibe o formulário de baixa de pagamento.
- O Usuário informa se o pagamento é total ou se é pago parcialmente.
- O Usuário insere o valor pago/recebido.

- O Usuário informa a forma de pagamento/recebimento.
- O usuário confirma a quitação da parcela e a janela do formulário é fechada.

Fluxo Alternativo

Gerar Parcela

- Pré-Condição: o Usuário acionou o comando "Pagar Parcialmente"
- Passos: O Sistema cria uma nova parcela não quitada com o valor restante a ser quitado.

Criar Desconto/Acréscimo

- Pré-Condições:
 - O Usuário informou o valor de pagamento diferente do valor da parcela.
 - O Usuário não selecionou a opção "Pagar Parcialmente"
- Passos: O Sistema interpreta a diferença de valor como desconto ou acréscimo.

Pagamento a Terceiros

- Pré-Condição: O Usuário acionou a opção "Adicionar Pagamento para Terceiros".
- Passos: O Sistema exibe um campo para preenchimento do nome do fornecedor desejado.

Enviar pagamento para CNAB Remessa

- Pré-Condições:
 - A parcela informada se refere a um pagamento.
- Passos:

- O **Sistema** exibe a opção para enviar pagamento via remessa.
- O Usuário seleciona a opção para pagamento via remessa.
- O **Sistema** disponibilizará a parcela para ser adicionada em um arquivo de remessa no módulo CNAB Remessa.

Caso de Uso: Visualizar Informações do Pagamento

Ator: Usuário.

Pré-Condições: O Usuário está autenticado no **Sistema** e possui permissão de acesso a esta opção de módulo.

Pós-Condições: O **Sistema** exibe um quadro com mais informações sobre a parcela desejada, tais como detalhamento da forma de pagamento (caso já quitada), situação da parcela, data de vencimento, data de pagamento, fornecedor, número de lançamento contábil.

Fluxo Básico:

- O Usuário seleciona os registros desejados.
- O Usuário aciona a opção "Visualizar" no menu superior

Caso de Uso: Mesclar Parcelas

Ator: Usuário

Pré-Condições:

- O Usuário está autenticado no **Sistema** e possui permissão de acesso a esta opção de módulo.

As parcelas a serem mescladas devem ser da mesma unidade de negócio e do mesmo cliente/fornecedor.

Pós-Condições: O Usuário pode mesclar diversas parcelas não quitadas desde que sejam da mesma unidade de negócio e do mesmo fornecedor/cliente para que a quitação do pagamento seja realizada em uma única operação.

Fluxo Básico:

- O Usuário seleciona as parcelas desejadas.
- O Usuário aciona a opção "Mesclar Pagamento".
- O **Sistema** exibe o formulário de quitação de parcelas.
- O Usuário preenche o valor pago/recebido e a multa referentes a cada parcela.
- O Usuário insere os números de série e notas fiscais no campo "Referência Nota Fiscal / N° de Serie".
- O Usuário confirma pagamento.
- O **Sistema** fecha a tela com o formulário de pagamento.

Criar Desconto/Acréscimo

- Pré-Condições:
 - O Usuário informou o valor de pagamento diferente do valor da parcela.
- Passos:
 - O **Sistema** interpreta a diferença de valor como desconto ou acréscimo.
 - O Usuário preenche a justificativa para a diferença de valore

Pagamento a Terceiros

- Pré-Condição: O Usuário acionou a opção "Adicionar Pagamento para Terceiros".

- Passos: O **Sistema** exibe um campo para preenchimento do nome do fornecedor desejado.

Enviar pagamento para CNAB Remessa

- Pré-Condições:
 - As parcelas informadas se referem a pagamentos.
- Passos:
 - O **Sistema** exibe a opção para enviar pagamento via remessa.
 - O Usuário seleciona a opção para pagamento via remessa.
 - O **Sistema** disponibilizará a parcela para ser adicionada em um arquivo de remessa no módulo CNAB Remessa.

- **Caso de Uso: Estornar Quitação de Pagamento**

Ator: Usuário

Pré-Condições:

- O Usuário está autenticado no **Sistema** e possui permissão de acesso a esta opção de módulo.
- As parcelas desejadas encontram-se quitadas.

Pós-Condições: A quitação de pagamento/recebimento é cancelada e a parcela é aberta novamente para quitação.

Fluxo Básico:

- O Usuário seleciona as parcelas desejadas.
- O Usuário aciona a opção "Estornar".
- O **Sistema** exibe tela de confirmação de estorno.

- O Usuário confirma o estorno
- O Sistema altera o status da parcela para "Aberta" a pagamento/recebimento.

- **Caso de Uso: Alterar data de Vencimento**

Ator: Usuário

Pré-Condições:

- O Usuário está autenticado no Sistema e possui permissão de acesso a esta opção de módulo.
- A parcela desejada não se encontra quitada.
- A parcela não foi enviada para remessa.

Pós-Condições: A data de vencimento da parcela será alterada.

Fluxo Básico:

- O Usuário seleciona a parcela desejada.
- O Usuário aciona a opção "Editar Vencimento".
- O Sistema exibe formulário para alterar a data de vencimento.
- O Usuário insere nova data de vencimento.
- O Usuário clica em "Alterar" para confirmar nova data.
- O Sistema altera a data de vencimento no banco de dados.

- **Caso de Uso: Alterar data de Quitação**

Ator: Usuário

Pré-Condições:

- O Usuário está autenticado no Sistema e possui permissão de acesso a esta opção de módulo.

- A parcela desejada se encontra quitada.
- A parcela não foi enviada para Remessa.

Pós-Condições: A data de pagamento da parcela será alterada.

Fluxo Básico:

- O Usuário seleciona a parcela desejada.
- O Usuário aciona a opção "Editar".
- O **Sistema** exibe o formulário para alterar a data de quitação.
- O Usuário insere nova data de quitação.
- O Usuário clica em "Alterar" para confirmar nova data.
- O **Sistema** altera a data de vencimento no banco de dados.

ANEXO 5: Casos de uso do módulo "Pagamento de Frete"

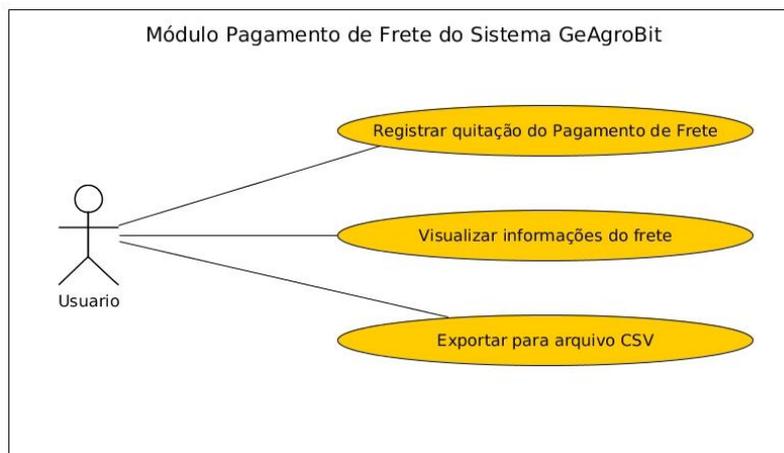


Diagrama de Caso de Uso do módulo "Pagamento de Frete".

Caso de Uso: Registrar Quitação de Pagamento de Frete

Ator: Usuário.

Pré-Condições:

- O Usuário está autenticado no **Sistema** e possui permissão de acesso a opção de módulo Baixar Pagamento.
- O frete a ser quitado está com o status "Não pago".

Pós-Condições: É registrada a quitação do frete selecionado, disponibilizando informações no módulo "Relatório de Pagamento de Frete".

Fluxo Principal:

- O Usuário seleciona o registro frete desejado.
- O Usuário aciona o comando "Baixar Pagamento".
- O **Sistema** exibe o formulário de baixa de pagamento.
- O **Sistema** preenche o formulário com os dados obtidos do módulo de origem do frete: unidade de negócio, fornecedor, número da nota fiscal, peso de saída e peso de chegada.
- O Usuário informa a unidade de negócio.
- O Usuário informa o título do frete.

- O Usuário informa a conta contábil de crédito.
- O Usuário informa o valor do produto e do frete por quilo.
- O Usuário informa a porcentagem de tolerância.
- O **Sistema** calcula o valor da tolerância de diferença de peso e exibe o resultado para o Usuário.
- O **Sistema** calcula o valor do desconto conforme a tolerância e exibe o resultado para o Usuário.
- O **Sistema** calcula o valor da fatura e exibe o resultado para o Usuário.
- O Usuário confirma a quitação do frete.
- O **Sistema** muda o status do pagamento para "pago".
- O **Sistema** fecha o formulário.
- O **Sistema** disponibiliza este pagamento em "Relatório de Pagamento de Frete".

Caso de Uso: Visualizar Informações do Frete

Ator: Usuário.

Pré-Condição: O Usuário está autenticado no **Sistema** e possui permissão de acesso a esta opção de módulo.

Pós-Condições: O **Sistema** exibe um quadro com mais informações sobre a parcela desejada, tais como detalhamento da forma de pagamento (caso já quitada), situação da parcela, data de vencimento, data de pagamento, fornecedor, número de lançamento contábil.

Fluxo Básico:

- O Usuário seleciona os registros desejados.
- O Usuário aciona a opção "Visualizar".

- O **Sistema** exibe as informações do frete: Resumo do contrato, número de ticket, fornecedor de frete, peso bruto, peso líquido, peso fardo, data de saída, placa do veículo, resumo de alterações na nota fiscal.

Caso de Uso: Exportar informações para arquivo CSV

Ator: Usuário.

Pré-Condições: O Usuário está autenticado no **Sistema** e possui permissão de acesso a esta opção de módulo.

Pós-Condições: O **Sistema** gera arquivo CSV para ser lido em sistema de planilhas eletrônicas, contendo os dados selecionados pelo usuário.

Fluxo Básico:

- O Usuário seleciona os registros desejados.
- O Usuário aciona a opção "Exportar CSV".
- O **Sistema** exibe os tipos de informações que podem ser exportadas.
- O Usuário seleciona as informações desejadas.
- O Usuário aciona a opção "Gerar CSV".
- O **Sistema** gera arquivo CSV para download.