

**CURSO DE SISTEMAS DE INFORMAÇÃO**

**IGOR GOMES DE MOISÉS**

**ESTRATÉGIA DE SOFTWARE NA TECNOLOGIA JAVA: AUTOMAÇÃO DE TRANSAÇÕES EM SGBD’S RELACIONAIS**

**FORTALEZA  
2017**

**IGOR GOMES DE MOISÉS**

**ESTRATÉGIA DE SOFTWARE NA TECNOLOGIA JAVA: AUTOMAÇÃO DE TRANSAÇÕES EM SGBD’S RELACIONAIS**

Projeto apresentado Curso de Sistemas de Informação do Centro Universitário Estácio do Ceará como requisito parcial para obtenção do título de Bacharel.

Orientador: Prof. Ms. Francisco Alves Carneiro

Coorientadora: Profa. Dra. Letícia Adriana Pires Ferreira dos Santos

**FORTALEZA  
2017**

M714e Moisés, Igor Gomes de

Estratégia de software na tecnologia java: automação de

transações em SGB’S relacionais Igor Gomes de Moisés

Silva. – Fortaleza, 2017.

60f.; 30cm.

Projeto apresentado ao curso de Sistemas de Informação, para a obtenção do grau de bacharel em Sistemas de Informação da Universidade Estácio de Sá.

1. Banco de Dados 2. Frameworks 3. Java I.Título II. Universidade Estácio de Sá.

CDD 005.74

**IGOR GOMES DE MOISÉS**

**ESTRATÉGIA DE SOFTWARE NA TECNOLOGIA JAVA: AUTOMAÇÃO DE TRANSAÇÕES EM SGBD’S RELACIONAIS**

Projeto apresentado Curso de Sistemas de Informação do Centro Universitário Estácio do Ceará como requisito parcial para obtenção do título de Bacharel.

Orientador: Ms. Francisco Alves Carneiro

Coorientadora: Profa. Dra. Letícia Adriana Pires Ferreira dos Santos

Aprovado em \_\_\_/\_\_\_/\_\_\_\_\_.

**BANCA EXAMINADORA**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Prof. Ms. Francisco Alves Carneiro (Orientador)

Centro Universitário Estácio do Ceará

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Profª Dra. Letícia Adriana Pires Ferreira dos Santos (Coorientadora)

Centro Universitário Estácio do Ceará

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Prof. Ms. Aminadabe Barbosa de Sousa

Centro Universitário Estácio do Ceará

Aos meus pais, João Hélio e Silmia.

A minha avó, Maria Neri Silva (*in memoriam*).

**AGRADECIMENTOS**

Agradeço, primeiramente, a minha falecida avó Maria Neri Silva, que com o ar de sua bondade me objetivou para que eu estudasse e escolhesse esta área de atuação profissional.

Aos meus falecidos avós, José Moisés e Marcelina Melo, a minha falecida tia e mãe por consideração, Celi Melo, que sempre me apoiaram e me incentivaram. Agradeço aos meus pais, João Hélio e Silmia, que me apoiam todos os dias, seja na vida pessoal e ou profissional, fazendo com que eu galgue sempre novos desafios.

A minha namorada, Joselane Lima, que me apoiou e me incentivou na criação deste trabalho.

Aos meus colegas de faculdade e profissão que participaram da pesquisa.

Ao meu amigo Fábio Rodrigues Franco Júnior que me auxiliou no desenvolvimento da ferramenta apresentada neste documento.

Ao meu orientador, Prof. Francisco Alves Carneiro, pelo suporte analítico prestado, culminando em correções e incentivos.

A minha coorientadora, Profa. Dra. Leticia Adriana Pires Ferreira dos Santos, que por meio da relevância do seu trabalho científico, possibilitou-me, no pouco tempo que lhe coube, a realização deste trabalho.

Aos professores que, gentilmente, aceitaram participar da banca de defesa do TCC.

E a todos que fizeram parte da minha formação, direta ou indiretamente, deixo aqui os meus mais sinceros votos de agradecimento.

“Especificar requisitos detalhadamente de modo que uma máquina possa executá-los é programar *–* e tal especificação é o código*”* (MARTIN, 2008, p.2).

**RESUMO**

Bancos de dados representam a base de armazenamento de informações de todo sistema. Estruturados logicamente são conjuntos de dados que, quando agrupados, definem um escopo único de caráter especializado e eficiente para consultas a informações. Para o gerenciamento dessa tecnologia foram criados os SGBD, Sistemas Gerenciadores de Bancos de Dados. Esses são *softwares* responsáveis pelo controle do banco de dados, assim como pela atuação de interface de entrada e saída de demandas. A tecnologia Java possui um conjunto de interfaces únicas para a realização da integração com SGBDs. Uma das etapas do trabalho de desenvolvimento de todo *software* é a montagem das estruturas de integração e realização de transações da tecnologia com o banco de dados, a qual termina por demandar uma certa quantidade de tempo do trabalho de programação. O objetivo deste trabalho é, portanto, apresentar a ferramenta SigmaDB, um novo Framework capaz de realizar a integração da tecnologia Java com qualquer SGBD relacional, buscando trazer agilidade de desenvolvimento, centralização da estrutura de códigos, boas práticas de desenvolvimento, assim como o controle e manutenção das tabelas do banco de dados. Utilizando-se dos conceitos avançados da tecnologia Java, a ferramenta foi desenvolvida baseada nas estruturas de *generics,* que lhe permite a manipulação de qualquer tipo de objeto, e da estrutura de reflexão, que lhe dá o poder de quebrar a regra de encapsulamento da linguagem de programação. Dessa forma, ao receber qualquer objeto, se este seguir a convenção de *getters* e *setters,* a ferramenta será capaz de manipulá-lo da forma como ela achar necessário. Diante disto, a SigmaDB é capaz de converter o paradigma de Orientação a Objetos em consultas a bancos de dados, trazendo os benefícios citados.

Palavras-Chave:Banco de dados. Frameworks. Java. Tuplas.

**ABSTRACT**

Databases represent the basis for storing system-wide information. Logically structured are datasets that, when grouped together, define a unique, specialized and efficient scope for querying information. For the management of this technology, DBMSs were created, Database Management Systems. These are software that is responsible for the control of the database, as well as for the interaction of input and output interface of demands. Java technology has a set of unique interfaces for integrating with DBMS. One of the steps in the development of all software is the assembly of the integration and transaction structures of the technology with the database, which ends up requiring a certain amount of programming time. The objective of this work is, therefore, to present the SigmaDB tool, a new framework capable of integrating Java technology with any relational DBMS, aiming to bring development agility, centralization of code structure, good development practices, as well as control and maintenance of database tables. Using the advanced concepts of Java technology, the tool was developed based on the structures of generics, which allows it to manipulate any type of object, and the reflection structure, which gives it the power to break the language encapsulation rule Of programming. In this way, upon receiving any object, if it follows the convention of getters and setters, the tool will be able to manipulate it in whatever way it deems necessary. In view of this, SigmaDB is able to convert the Object Orientation paradigm into database queries, bringing the mentioned benefits.

Keywords: Database. Frameworks. Java. Tuples.

**SUMÁRIO**

[1 INTRODUÇÃO 11](#_Toc482301838)

[2 PROPOSTA DE TRABALHO 14](#_Toc482301839)

[2.1 Metodologia de Trabalho 16](#_Toc482301840)

[2.2 Precisão de Alocação de Recursos 17](#_Toc482301841)

[2.3 Cronograma de trabalho 18](#_Toc482301842)

[3 A EMPRESA E O NEGÓCIO 19](#_Toc482301843)

[3.1 Histórico da empresa 19](#_Toc482301844)

[3.2 Atividades da empresa 19](#_Toc482301845)

[3.3 Organograma 20](#_Toc482301846)

[3.4 Mercado consumidor 20](#_Toc482301847)

[3.5 Concorrência 21](#_Toc482301848)

[3.6 Premissas e Restrições ao projeto na organização 21](#_Toc482301849)

[4 O SISTEMA ATUAL 24](#_Toc482301850)

[4.1 Justificativa de Escolha do Sistema 24](#_Toc482301851)

[4.1.1 O Sistema 24](#_Toc482301852)

[4.1.2 Funcionamento do Sistema 29](#_Toc482301853)

[*4.1.2.1 Configurando o Framework 29*](#_Toc482301854)

[*4.1.2.3 Tratamento de tabelas 31*](#_Toc482301855)

[4.1.3 O Ambiente do Sistema 39](#_Toc482301856)

[4.1.4 A definição do escopo 39](#_Toc482301857)

[4.2 Motivação para o novo sistema 40](#_Toc482301858)

[4.3 Situação desejada 40](#_Toc482301859)

[5 O SISTEMA PROPOSTO 41](#_Toc482301860)

[5.1 Lista de Requisitos do Sistema 41](#_Toc482301861)

[5.1.1 Requisitos Funcionais 41](#_Toc482301862)

[5.1.2 Requisitos Não Funcionais 43](#_Toc482301863)

[5.2 Diagrama de Caso de Uso 44](#_Toc482301864)

[5.3 Especificações dos casos de uso 45](#_Toc482301865)

[5.3.1 Configura ferramenta 45](#_Toc482301866)

[5.3.2 Persistência de dados 46](#_Toc482301867)

[5.3.3 Pesquisa de dados 46](#_Toc482301868)

[5.3.4 Persistência de logs 48](#_Toc482301869)

[5.3.5 Pesquisa de logs 49](#_Toc482301870)

[5.4 Modelo Conceitual de Classes 51](#_Toc482301871)

[5.5 Modelo Conceitual de Dados 52](#_Toc482301872)

[6 CONCLUSÕES 53](#_Toc482301873)

[REFERÊNCIAS 54](#_Toc482301874)

# 1 INTRODUÇÃO

O desenvolvimento de um software de qualidade requer atenção para um conjunto grandioso de variáveis. Custo, qualidade, tempo, modelos de processos, arquitetura operacional, dentre outros, são exemplos de vertentes que devemos nos preocupar ao criar um sistema computacional. Assegurando tudo isso, por trás do software, existe a necessidade de um arcabouço capaz de manter todo o conjunto de informações trabalhadas.

“Um sistema de banco de dados é uma coleção de dados inter-relacionados e um conjunto de programas que permitem aos usuários acessar e modificar esses dados” (SILBERSCHATZ; KORTH; SUDARSHAN, 2006, p.2). Além dos conjuntos de programas que compõe um banco de dados, esse também pode ser mantido por meio de outras aplicações, denominadas de SGBD (Sistemas Gerenciadores de Bancos de dados). “Um sistema gerenciador de banco de dados (DBMS – Database Management System) é uma coleção de programas que permite aos usuários criar e manter um banco de dados” (ELMASRI; NAVATHE, 2011, p.3).

Segundo Deitel e Deitel (2010), um sistema de gerenciamento de banco de dados fornece mecanismos para armazenar, organizar, pesquisar e alterar informações para muitos usuários. Os SGBD’s disponibilizam acesso de dados sem que a representação interna desses sejam expostas para os usuários.

Elmasri e Navathe (2011) apontam que bancos de dados desempenham um papel crítico em quase todas as áreas que se utilizam de computadores para o controle de suas necessidades. Os *softwares*, também em sua maioria, necessitam do banco de dados para que possam passar a armazenar de forma eficiente e segura todas as informações que lhe são pertinentes.

O desenvolvimento de um software se dá por meio da escrita de códigos fonte numa determinada linguagem de programação. Uma das linguagens de programação mais difundida e utilizada atualmente no mercado de trabalho é o Java. Segundo o site Business Insider, a linguagem de programação Java no ano de 2016 ficou com a segunda colocação de tecnologia de desenvolvimento mais utilizada no mundo, perdendo apenas para o Javascript.

Segundo sua documentação, o Java possui uma base bem estruturada para o desenvolvimento de praticamente todos os tipos de aplicações em rede, assim como é o padrão global para o desenvolvimento e distribuição de aplicações móveis, jogos, conteúdo baseado na *Web* e *softwares* corporativos. Isso deve-se ao fato da sua arquitetura ser mantida por uma máquina virtual chamada de JVM (*Java Virtual Machine*) e muitas APIs de controle de desenvolvimento.

A JVM traz uma característica única ao Java, sua portabilidade sistêmica. Ela é um programa que carrega e executa aplicações Java, traduzindo *bytecodes* em código executável. Popek e Goldberg (1974) nos dizem que uma máquina virtual é uma cópia isolada de uma máquina real. Exatamente por conta desta característica, o Java torna-se escalável suficiente para ser executado em qualquer sistema operacional.

No desenvolvimento de aplicações corporativas, existe a necessidade de se realizar a integração da tecnologia utilizada para o desenvolvimento com algum SGBD, tendo em vista que essa é a forma mais segura e otimizada para o armazenamento.

Tomando como base a tecnologia Java, existem dois protocolos distintos que servem para a realizar a integração do Java com SGBDs. Esses protocolos são ODBC (*Open Database Connectivity*)*,* um padrão de acesso para várias linguagens de programação por meio de *drivers* que se transforma numa camada de tradução entre a linguagem de programação e o SGBD, e o JDBC (*Java Database Connectivy*), que é uma API Java que permite que aplicações se conectem a um banco de dados, sendo esse o protocolo mais utilizado pela comunidade para essa finalidade.

Por meio de ambos os protocolos, a tecnologia Java possui estruturas de códigos, denominadas de classes e interfaces, que são disponibilizadas em suas APIs nativas (pacote java.sql.\*), que são responsáveis pela comunicação da linguagem com um banco de dados.

O conhecimento adquirido durante o tempo de oito anos de trabalho profissional com a tecnologia Java, me permitiu projetar e desenvolver uma ferramenta capaz de automatizar transações da tecnologia Java com o banco de dados, assim como centralizar todas as conexões num único ponto de acesso.

Desenvolvida inicialmente para atender demandas de uma *startup* de minha propriedade, essa ferramenta foi desenvolvida tomando como base os conceitos de reflexão, que tem o papel de realizar a quebra de encapsulamento da linguagem de programação e da estrutura de *generics* da API Java, com o objetivo de conseguir tratar qualquer conteúdo a ser trabalhado pelo programador. Dessa forma, foi possível desenvolver uma estrutura de centralização de código, criando uma área de alta coesão que pode ser reutilizada em qualquer projeto.

Do ponto de vista do trabalho de programação, a parte inicial de um projeto demanda uma determinada quantidade de tempo para a construção das estruturas de integração da aplicação que está sendo trabalhada com sua respectiva base de dados. O *software* apresentado neste documento tem como objetivo fazer com que esta etapa do projeto esteja automaticamente concluída, uma vez que a ferramenta proposta é uma ponte que interliga o Java a qualquer banco de dados. Possui também o intuito de buscar trazer coesão aos códigos do projeto, uma vez que a ferramenta será a controladora de qualquer transação com o banco.

Este trabalho inicia-se com uma breve apresentação sobre seu escopo por meio de uma introdução, localizada no capítulo um. Essa tem como objetivo principal dar iniciação ao objetivo principal que é a ligação de bancos de dados com a tecnologia Java.

Dada a introdução, o próximo ponto é a narrativa de apresentação da ferramenta, visando apontar sua finalidade, assim como seus objetivos. Encontraremos também a forma pela qual todo o conteúdo aqui apresentado foi trabalhado, incluindo toda a alocação de recursos necessárias e os métodos de pesquisa utilizados para a elaboração deste documento. Todas essas características serão apresentadas por meio da proposta de trabalho, que poderá ser encontrada no capítulo dois do documento.

Conforme mencionado, a ferramenta foi criada com o propósito de suprir a necessidade de uma *startup* criada pelo autor deste trabalho. No terceiro capítulo será apresentado uma visão da empresa detentora da *startup*, apresentado seu histórico, mercado, listagem de serviços, estrutura funcional e como esta ferramenta se encaixa e traz melhorias para os objetivos da organização.

O quarto capítulo destina-se a apresentar o funcionamento da ferramenta, bem como a parte técnica do documento, mostrando como de fato o programador deve realizar seu trabalho com a ferramenta.

No quinto capítulo, encontra-se como o sistema foi modelado. Apresenta-se, ainda, toda a análise de projeto realizada para a construção do produto.

## 2 PROPOSTA DE TRABALHO

Para o desenvolvimento de qualquer aplicação se faz necessário a existência de uma camada de armazenamento de dados em disco. Atualmente, a área da computação considera que a utilização de bancos de dados é a maneira mais segura para a realização do armazenamento de informações.

Segundo Elmasri e Navathe (2011), banco de dados é uma cadeia de informações relacionadas. Essas informações são fatos conhecidos que podem ser registrados e possuem significado implícito. Para o trabalho com banco de dados existem os SGBDs (Sistemas gerenciadores de bancos de dados), que são *softwares* que possuem recursos capazes de manipular as informações do banco de dados e interagir com usuários. Assim sendo, podemos citar como exemplo de SGBDs os *softwares* Oracle, PostgreSQL, MySQL, SQL Server, MongoDB, entre outros.

Bancos de dados possuem modelos lógicos de implementação. O mais difundido no mercado atualmente é o modelo relacional, inventado por Edgar Fran Codd (1970) e mantido por Chris Date e HugDarwen como um modelo geral de dados. Esse modelo é considerado o mais seguro e mais rápido, em face do modelo permitir ser estendido para atender as características de orientação a objeto e por estar fundamentado num modelo matemático, oriundo da Teoria dos Conjuntos e da Lógica dos Predicados de Primeira Ordem.

Este modelo é baseado no conceito de tabelas, onde cada tabela possui suas respectivas linhas, denominadas como tuplas, e colunas, denominadas como atributos. Todo Sistema Gerenciador de Banco de Dados, que implementa este modelo, deve possuir uma estrutura capaz de aceitar determinados tipos de comandos para o gerenciamento do banco de dados. Esses comandos são definidos por meio de uma linguagem denominada de SQL e segue agrupamentos definidos como DDL (*Data Definition Language*), DML (*Data Manipulation Language*), DCL (*Data Control Language*) e TCL (*Transaction Control Language*).

Navathe (2011) afirma que o nome SQL é uma sigla para *Structured Query Language* (Linguagem de Consulta Estruturada) e que esta pode ser considerada um dos principais motivos para o sucesso dos bancos de dados relacionais comerciais.

Toda tecnologia de linguagem de programação de alto nível possui sua interface de conexão com bancos de dados, mesmo que esta tenha sua implementação auxiliada pelo próprio sistema operacional.

Tratamos, em específico, neste trabalho sobre a tecnologia Java, tendo em vista que a ferramenta de trabalho aqui proposta foi modelada e implementada nesta plataforma.

Para a realização da conexão do SGBD com a tecnologia Java, existem dois protocolos distintos de padrões de acesso que podem ser seguidos. O profissional da área de programação pode optar pelo protocolo ODBC (*Open Database Connectivity*)*,* um padrão de acesso para várias linguagens de programação por meio de *drivers* que se transforma numa camada de tradução entre a linguagem de programação e o SGBD. Em paralelo, o profissional da área de programação também pode optar pelo protocolo JDBC, um conjunto de códigos Java responsável por realizar a comunicação da linguagem com qualquer SGBD que tenha o modelo lógico relacional.

A linguagem de programação Java necessita de classes especiais para a realização da conexão com o SGBD. Essas classes especiais encontram-se acopladas no pacote java.sql\* de sua API pública. O desenvolvimento lógico para tratar da conexão da tecnologia com o SGBD geralmente envolve uma longa estrutura de código, o que sempre acaba demandando uma determinada quantidade de tempo nos projetos.

Com o intuito de evitar o retrabalho dos desenvolvedores, de tornar mais rápidas as implementações de código, de buscar trazer melhor entendimento do conceito lógico do sistema, de unificar soluções e de dar coesão aos códigos fontes, grupos de desenvolvimento de código fonte livre passaram a implementar ferramentas que automatizam conexões e comandos SQL ao SGBD. A essas ferramentas, generalizamos, chamando-as de *frameworks* de bancos de dados. Podemos citar, como exemplo, a ferramenta Hibernate e JPA, que são ferramentas mundialmente difundidas para integração da tecnologia Java com banco de dados.

A ideia deste trabalho é, assim, apresentar uma ferramenta desenvolvida pelo autor, que tem o papel de realizar a conexão da linguagem de programação Java com qualquer banco de dados, realizando a execução de qualquer requisição ao banco, sem que seja necessário a escrita de comandos SQL.

Por meio do paradigma de Orientação a Objeto, a ferramenta é capaz de interpretar qualquer tipo de requisição embutidas em objetos e destiná-la em forma de uma transação ao SGBD, o qual irá processar a respectiva requisição e respondê-la a ferramenta, que por consequente entregará o retorno ao solicitante. Não somente isto, a ferramenta também é capaz de executar comandos SQL, obtendo-se o mesmo comportamento descrito anteriormente.

Tomando como base a aplicação desta ferramenta numa empresa, podemos observar os seguintes benefícios promovidos por uma única estrutura:

* **Conexão automática com qualquer banco de dados** – A ferramenta é capaz de realizar a conexão da linguagem Java com qualquer banco de dados, por intermédio das bibliotecas configuradas no projeto, e por meio de um arquivo de propriedades que deve ser entregue a ela. Nesse arquivo deverá encontrar-se as informações necessárias para conexão com o banco. A saber, login, senha, *driver* e URL do banco**;**
* **Único ponto de entrada e saída e remoção de dados** – A ferramenta oferece uma estrutura única para a realização de comandos de persistência de dados no banco, assim como para pesquisa de dados do banco. Isso faz com que não exista redundância de código na implementação de consultas e persistências;
* **Não necessita de SQL** – Através de uma estrutura genérica da O.O, baseada na metodologia de reflexão para quebra de encapsulamento de objetos, a ferramenta é capaz de interpretar objetos e gerar o respectivo código SQL e executá-lo internamente, obtendo o resultado para o usuário;
* **Executa SQL** - Caso ainda seja necessário a realização da execução de um código SQL, a ferramenta é capaz de executar qualquer sintaxe SQL, retornando o resultado da mesma maneira;
* **Velocidade de desenvolvimento** – Para a realização da integração da ferramenta com um Banco de dados relacional, basta apenas que o programador configure, por meio de um arquivo XML, alguns parâmetros indicando a URL do banco, driver, usuário e senha. A partir daí, por meio da utilização do paradigma de O.O, já é possível a realização de qualquer transação por intermédio da ferramenta.

## 2.1 Metodologia de Trabalho

Este trabalho foi elaborado por meio dos seguintes métodos:

* Pesquisa bibliográfica em livros relacionados a linguagem de programação Java, em livros relacionados a padrões de projetos e em livros relacionados a boas práticas de programação;
* Pesquisa em sites na internet;
* Ajustes de código para otimização da ferramenta, baseado nas consultas realizadas.

A pesquisa bibliográfica é definida por Koche (2006) como sendo a que se desenvolve procurando explicar um problema, utilizando o conhecimento disponível, a partir das teorias evidenciadas em livros ou obras congêneres. Na pesquisa bibliográfica, o investigador levanta o conhecimento disponível na área, identificando as teorias produzidas, analisando-as e avaliando a contribuição delas para auxiliar a compreender ou explicar o problema objeto da investigação.

## 2.2 Precisão de Alocação de Recursos

Para o desenvolvimento do Framework, foi necessário que os recursos humanos e materiais fosse respectivamente alocados da seguinte maneira:

* **Recursos Humanos** – Autor do projeto;
* **Recursos Materiais** – Computador MacBook Pro, hospedagem de serviço de banco de dados em nuvem, hospedagem de sistema *web*, *softwares* Ecplise e PostgresSQL.

Segundo Kotler (1998), o planejamento estratégico é, dessa forma, definido como um processo gerencial de desenvolver e manter um ajuste viável entre os objetivos, as habilidades, os recursos de uma organização e as oportunidades de um mercado em constante mudança.

**Tabela 1 - Recursos de Hardware e Software**

|  |  |  |
| --- | --- | --- |
| Quantidade | Recurso | Custo |
| 1 | Computador MacBook Pro | R$ 6000,00 |
| 1 | Hospedagem de servidor de banco de dados | R$ 240,00 |
| 1 | Hospedagem de servidor com container de aplicação Tomcat | R$ 240,00 |
| 1 | Softwares para desenvolvimento de sistemas Navicat | R$ 400,00 |
| 1 | Software Eclipse e PostgreSQL | R$ 0,00 |

Fonte: Elaborada pelo autor.

**Tabela 2 - Recursos Humanos**

|  |  |  |
| --- | --- | --- |
| Quantidade | Recurso | Custo |
| 1 | Analista de Sistemas Sênior com habilidade em programação Java – 2600 horas | R$ 15.000,00 |

Fonte: Elaborada pelo autor.

## 2.3 Cronograma de trabalho

O planejamento das atividades no tempo adequado para alcançar os objetivos pode ser verificado na Tabela 3:

**Tabela 3 - Cronograma de Trabalho**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Fevereiro | Março | Abril | Maio |
| Proposta do Trabalho | X |  |  |  |
| Estabelecimento dos Objetivos | X |  |  |  |
| Leitura da bibliografia | X | X | X | X |
| Parte 1 do trabalho, documentação parcial do trabalho. (Itens 1, 2 e 3) | X | X | X |  |
| Trabalho final com documentação completa do trabalho. (Itens 1, 2 e 3 revisados e 4 e 5) |  |  | X | X |

Fonte: Elaborada pelo autor.

## 3 A EMPRESA E O NEGÓCIO

A Cientz Publicidade e Tecnologia é uma empresa que atua há 10 anos no mercado de publicidade e propaganda na cidade de Fortaleza, no estado do Ceará. Atualmente é detentora de *startups* na área de inovação de projetos em tecnologia da informação.

## 3.1 Histórico da empresa

Com o intuito de entrar no neste novo mercado, a Cientz Publicidade e Tecnologia visa mesclar aos seus sistemas de informação a área da publicidade, aproximando ainda mais fornecedores e consumidores, de maneira que as oportunidades de mercado sejam ainda mais difundidas.

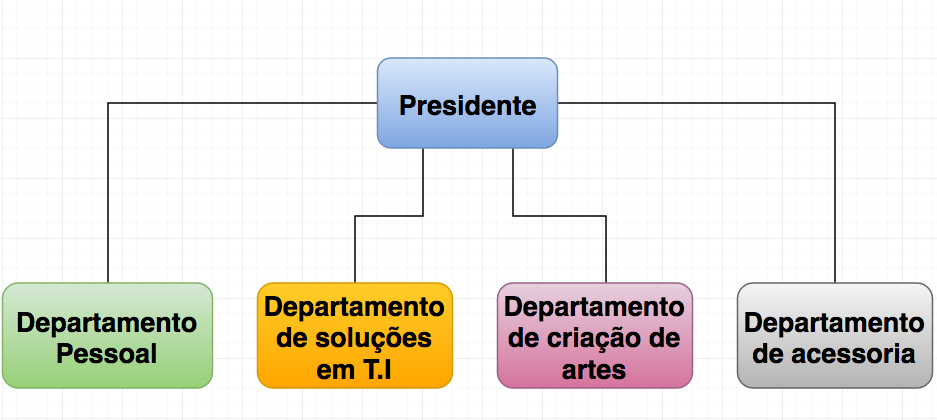
Sua proposta é que com um atendimento diferenciado, a empresa se coloque o mais próximo possível de seus clientes, de maneira que dessa forma o entendimento e atendimento de suas necessidades seja supridos o mais rápido possível.

## 3.2 Atividades da empresa

A Cientz Publicidade atua nas seguintes vertentes de serviço:

* Assessoria de Imprensa;
* Lançamento de Empresa;
* Lançamento ou Relançamento de Produto;
* Lançamento Imobiliário;
* Criação de *Briefing*;
* Desenvolvimento de Logotipo e Identidade Visual;
* Produção Digital;
* Monitoramento e Criação de Conteúdo para Mídias Sociais;
* Campanha Incentivo e/ou Relacionamento;
* Ação e Campanha Promocional;
* Portfólio;
* Plano de mídia;
* Levantamento de Dados sobre Cliente e/ou Produtos;
* Desenvolvimento de Estratégias de comunicação Publicitária;
* Criação de Material Promocional/Ponto de Venda;
* Apresentação em Feiras, Eventos e Convenções;
* Rotulagem e Etiquetas;
* Criação de Catálogos, Anúncios em Jornal, Adesivos, *Blimp*, *Busdoor*, *Display*, *Front Ligth*, Luminoso, *Outdoor*, Painéis, Placas *Backdrop*, Adesivação de Veículos e Outros;
* Desenvolvimento de *Startups*.

## 3.3 Organograma

****

**Figura 1 - Organograma da empresa**

Fonte: Elaborado pelo autor.

## 3.4 Mercado consumidor

A Cientz Publicidade não possui um público alvo a ser atingido, de maneira que qualquer empresa ou pessoa física que sentir a necessidade de um de seus serviços ofertados pode vir a consumi-los. Entretanto dentre seus maiores clientes estão as empresas Grupo Nova Distribuidora e Vent7 inovações, para os quais a empresa possui algumas *startups* na área de tecnologia da informação em desenvolvimento.

## 3.5 Concorrência

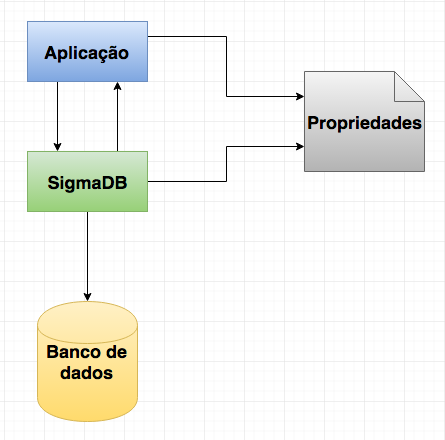
A Cientz Publicidade, se enquadra numa empresa de pequeno /médio porte, de forma que seu CEO Edison Araújo considera que existem inúmeras outras empresas do mesmo tamanho, ou profissionais informais trabalhando na área publicitária, o que, de certa forma, pode ser considerado como empresas concorrentes.

## 3.6 Premissas e Restrições ao projeto na organização

Atualmente a Cientz trabalha com as tecnologias de desenvolvimento PHP e Java. Como a ferramenta SigmaDB é desenvolvida exclusivamente para Java, os projetos da empresa, implementados nesta tecnologia, serão todos adaptados para que contemplem essa nova estrutura.

Por se tratar de um *framework* de transação de dados, a ferramenta pode ser utilizada para o desenvolvimento de aplicações de ambientes *desktops* e *web*.

Para a utilização da ferramenta, se fará necessário que os projetos da empresa sigam a arquitetura apresentada abaixo:



**Figura 2 - Arquitetura do Framework**

Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.

**Figura 3 - Diagrama de classes e relacionamentos do Framework**

Podemos observar que o *framework* é uma ponte direta que liga a comunicação da aplicação a ser desenvolvida com sua base de dados. Entre eles, além do relacionamento entre classes Java, existe também um arquivo de propriedades, responsável por armazenar as informações pertinentes a conexão com o banco de dados. Este arquivo de propriedades deve estar configurado no projeto da aplicação de acordo com as especificações do *schema* XSD da API da ferramenta.

Segundo Elmasri e Navathe (2011), uma entidade é equivalente a um objeto ou conceito do mundo real. Os autores complementam dizendo que um atributo representa alguma propriedade de interesse que compõe melhor uma entidade. Assim sendo, podemos concluir como entidade toda e qualquer tabela do banco de dados, assim como atributo as colunas destas tabelas.

Para o funcionamento da ferramenta, se faz necessário a existência de entidades do banco em formato de classe Java. Ou seja, se necessita de que exista um espelho da tabela em forma de um arquivo Java. Para cada tabela que o usuário programador necessite manter no banco, deverá haver uma classe Java com o mesmo nome da entidade, mesmos nomes de atributos e respectivos tipos de dados. Essa classe que representa uma entidade, obrigatoriamente deverá herdar a classe TableMaster da ferramenta, conforme podemos observar na figura 4. O atributo que representa uma Primary Key, deverá obrigatoriamente receber uma *annotation* PkTableMaster. Será por meio dessa interface que a ferramenta será capaz de identificar qual dos atributos da classe representarão a chave primária da entidade e desta forma montar as devidas restrições das transações.

Toda entidade tem um ponto único de entrada e saída de dados, que será a classe SigmaDB. Esta classe é capaz de realizar *inserts, updates, deletes* e *selects*. Os valores dos atributos da classe servirão como restrições para realização de consultas e persistência de dados.

Caso seja necessário a realização de consultas que envolva a junção de duas ou mais tabelas, o usuário deverá informar uma classe que delega as informações das entidades solicitadas, de maneira que a própria ferramenta se responsabilizará pela montagem da estrutura SQL das entidades contidas na classe, retornando uma lista de instâncias da classe informada contendo o resultado da consulta.

## 4 O SISTEMA ATUAL

Atualmente a empresa não possui nenhum *framework* de trabalho para tratar da comunicação da tecnologia Java com banco de dados. Toda a estrutura de conexão com banco é feita por meio de conexões JDBC e por meio de classes proprietárias aleatórias da própria empresa. Isto tem causado grande dificuldade de manutenção de códigos, devido ao fato de que a falta de administração dessas estruturas acabou por gerar códigos redundantes e estruturas completamente desarticuladas.

## 4.1 Justificativa de Escolha do Sistema

A escolha por elaborar e implantar o *framework* de transações com banco de dados, se dá pela necessidade de uma ferramenta mais robusta, veloz, centralizadora, que seja capaz de realizar qualquer comando SQL, sem que se faça necessário a escrita dos mesmos. Ou seja, uma ferramenta capaz de automatizar a conversão do paradigma O.O em estrutura SQL.

Existe a necessidade de se conseguir códigos coesos, desacoplados e não redundantes, de maneira que o profissional da área de desenvolvimento de *software* consiga agilidade e qualidade no seu trabalho.

## 4.1.1 O Sistema

A tecnologia Java possui uma estrutura nativa, dentro de sua API, para realizar conexões e transações com bancos de dados. Esta estrutura encontra-se localizada dentro do pacote de classes java.sql.\*. Dentro desse pacote podemos destacar como as principais entidades para a realização de conexão e comunicação com o banco como *DriverManager, Connection, Statement, PreparedStatement, ResultSet* e *RowSet*.

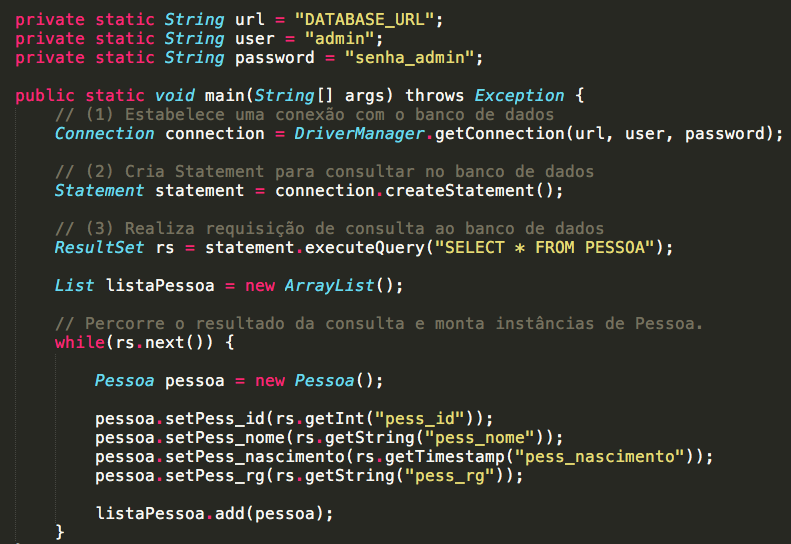
Tomando como base a imagem 8, para se realizar uma consulta a tabela Pessoa, por meio de um sistema desenvolvido na tecnologia Java, internamente o *software* deve seguir o conjunto de passos descritos adiante com os respectivos códigos fonte.

1. Por meio da entidade *DriverManager*, o programador deverá fazer com que o sistema abra uma conexão com o banco de dados. É neste momento que ele repassa as informações de URL, usuário e senha do banco de dados. A abertura dessa conexão resulta num objeto instanciado do tipo *connection*;
2. Com a conexão criada e aberta, representada pela instância de *connection*, o programador deverá fazer com que o sistema crie uma nova estrutura que seja capaz de receber demandas SQL. Assim sendo, a instância de *connection* é capaz de dar uma nova instância de *Statement* ou *PreparedStatement*, que são entidades aptas a receber sintaxe SQL;
3. Em posse da instância de uma dessas entidades, o programador deverá inserir a sintaxe SQL da requisição ao banco, ao mesmo tempo em que já envia a solicitação de consulta ao banco. A sintaxe SQL é dada ao programador, e ele sabe que obrigatoriamente neste determinado ponto uma sintaxe SQL única e predeterminada deverá ser utilizada.
4. O Java espera como retorno da consulta uma instância que é referenciada pela entidade *ResultSet*. Essa instância trabalha como uma tabela, que na verdade é um mapa de elementos de linhas e colunas. O programador deverá saber obrigatoriamente quais serão os atributos desse mapa assim como o seu respectivo tipo de valor. Conforme já explanado anteriormente, um atributo representa uma coluna de uma tabela do banco. Assim sendo, no *ResultSet* haverá todas as colunas, pedidas na consulta, da tabela Pessoa com cada valor de atributo em seus respectivos tipos.
5. Com o retorno da consulta em mãos, o programador deverá percorrer o *ResultSet* captando cada valor de atributo contido nele. De posse de todos os valores de atributos, que juntos representam uma tupla da tabela, é possível montar uma instância de pessoa de forma que essa por sua vez represente no Java uma tupla da tabela.

A priori, uma consulta numa única entidade não aparenta ser algo complexo e demorado, conforme podemos ver na figura 5. Entretanto, a complexidade da base de dados de um sistema não se resume somente numa única tabela, assim como numa única consulta. Então podemos levantar algumas desvantagens ao se trabalhar com o modelo apresentado na figura 5.

1. Se uma base de dados possuir quatrocentas tabelas distintas, seguindo o modelo apresentado, eu terei que utilizá-la quatrocentas vezes distintas respectivamente para cada tabela.
2. Se houver alguma alteração na tabela, a manutenção de código do sistema será muito grande, pois deverá ser alterada todas as consultas que mencionam essa tabela, todos os pontos que espelham esta tabela, etc.
3. Para consultas que utilizarão *joins* entre tabelas, se fará necessário a criação de uma nova estrutura, igual a apresentada acima, para a montagem das instâncias que representarão um registro da consulta realizada.
4. A maneira de se consultar e construir as instâncias ainda podem piorar considerando-se que a cada consulta que se utilizar de *joins* entre tabelas, será necessário uma nova estrutura igual a apresentada acima para a montagem das instâncias que representarão um registro da consulta realizadas.

**Figura 4 - Código para consulta da entidade Pessoa**



Fonte: Elaborado pelo autor.

Podemos perceber que se a cada consulta que o sistema necessitar realizar, o programador tenha que ter toda esta demanda de trabalho, haverá uma determinada quantidade de tempo que podemos considerar elevada e desnecessária do ponto de vista lógico. Torna-se um trabalho longo, árduo e de altíssima complexidade, pois uma manutenção em estruturas como essas tendem sempre a criar novos *bugs*. A visão para mapear todos os trechos de código do sistema que fazem menção para uma determinada tabela que foi alterada é de alta complexidade, pois alguns pontos podem ficar desatualizados.

Além disso, a visão que este modelo de trabalho traz é a de que inúmeras consultas diferentes poderão ser criadas de maneira descentralizada, resultando estruturas desacopladas que, por consequência, causam ainda mais dificuldade de manutenção do código.

Da mesma maneira que apresentamos a visão de uma consulta, podemos apresentar a visão de persistências na base de dados. Ainda tomando como exemplo a situação colocada acima, se para cada tabela do sistema, o programador tiver que mapear *inserts, updates* e *deletes*, com suas respectivas sintaxes SQL, também se tornará um trabalho árduo e demorado.

Por este tipo de necessidade, a comunidade de desenvolvimento do mundo Java começou a lançar ferramentas para minimizar e otimizar as transações com o banco de dados, de forma que este modelo de trabalho se torne desnecessário.

A SigmaDB é uma ferramenta estruturada sobre a tecnologia Java capaz de se integrar a qualquer SGBD relacional e realizar qualquer tipo de comando SQL dos tipos DML e DDL. Seu objetivo é ser uma ponte única de acesso ao banco realizando qualquer tipo de transação demandada pelo sistema.

A ferramenta, baseada na visão de reflexão do Java, é capaz de realizar a quebra do encapsulamento de instâncias e desta forma consegue converter o paradigma de O.O em sintaxes SQL cuja restrições serão os valores dos atributos contidos nas instâncias informadas a ferramenta.

Além dessa capacidade, ela traz recursos essenciais para o trabalho com as transações, podendo desta maneira o usuário optar por realizar o *commit* ou *rollback* de operações com o banco, a seu critério.

A ferramenta também é capaz de agregar ao *software* que a utiliza, uma estrutura de auditoria de segurança de tuplas, de maneira que um administrador identifique quem, quando e aonde um usuário realizou determinada persistência na base de dados. Em consequência desse benefício, o administrador terá em mãos o versionamento de cada registro das tabelas, graças a estrutura de auditoria gerada pelo *framework*. Essa estrutura de *logs* de registros faz com que o usuário seja capaz de realizar o *rollback* de toda uma transação, independentemente de qual data esta tenha ocorrido na vida funcional de um sistema. Essa é uma estrutura adicional da ferramenta, que pode ser ativada ou desativada por meio do arquivo de configuração.

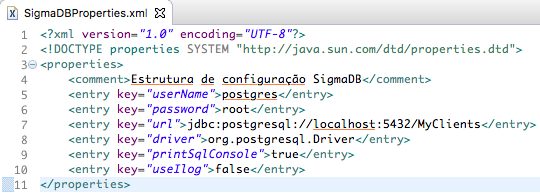
Com o intuito de simplificar ao máximo a utilização do *framework*, a configuração do mesmo se dar apenas pela criação de um arquivo do tipo XML no projeto onde será utilizado. Definidas todas as propriedades obrigatórias neste arquivo, a ferramenta já estará funcional. Ou seja, toda a conexão com SGBD já se encontra montada.

## 4.1.2 Funcionamento do Sistema

Mediante uma interface única de acesso, a ferramenta é capaz de realizar qualquer requisição a um SGBD. Seu objetivo é ser um ponto universal de passagens de requisições, de forma que pode ser instanciada em qualquer ponto da codificação sem que seja afetado o acoplamento das estruturas de código.

## *4.1.2.1 Configurando o Framework*

O *framework* deve ser configurado no projeto inicialmente incluindo a biblioteca de nome SigmaDB.jar no *classpath* do mesmo. Dentro da pasta “source”, onde ficarão alocados os pacotes de classes Java do projeto, deverá ser criado um arquivo de extensão XML avulso de nome SigmaDBProperties, mostrado na figura 5. Ele guardará o nome do usuário, senha, descrição de *driver*, URL de conexão, verificação de permissão de utilização do console da aplicação para escrever comandos SQL executados e a liberação para a utilização da estrutura de *log* e versionamento de registros das tabelas.



**Figura 5 - Visão do arquivo SigmaDBProperties.xml**

Fonte: Elaborado pelo autor.

#### 4.1.2.2 Definição de padrão de utilização

Visando excelência no desenvolvimento, a ferramenta foi construída baseada em padrões de projeto. Segundo Alexander, Ishikawa e Silverstein, (1977), um padrão descreve um problema que pode ocorrer inúmeras vezes em determinado contexto, e descreve ainda a solução para esse problema, de maneira que essa solução possa ser utilizada sistematicamente em situações semelhantes.

A ferramenta baseia-se num padrão conhecido e difundido de nome DAO (*Data Access Object*). Esse padrão visa definir uma interface única para a realização de processos que envolvam ligações diretas com um banco de dados. Desta maneira, torna-se viável a separação da regra de negócio das aplicações com as regras de acesso a banco de dados.

A ferramenta exige que alguns pontos de restrições de implementação sejam adotados na codificação do sistema.

* Toda tabela do banco de dados deve ter uma classe correspondente que tenha o mesmo nome e mesmos atributos com respectivos tipos;
* As classes que são espelhos de tabelas devem seguir o padrão *get* e *set*.

Toda classe do sistema que necessitar de uma interface com o banco de dados, deverá obrigatoriamente criar uma instância da classe SigmaDB que estará embutida dentro do *framework*. A figura 7 apresenta um exemplo da utilização de consulta e persistência de dados por meio da interface SigmaDB, assim como a listagem das principais funcionalidades disponibilizadas por ela. A classe SigmaDB é uma interface centralizadora de conexão com o banco, de maneira que ela controla toda e qualquer requisição feita ao mesmo.

**Figura 6 - Principais funcionalidades da classe SigmaDB**



Fonte: Elaborado pelo autor.

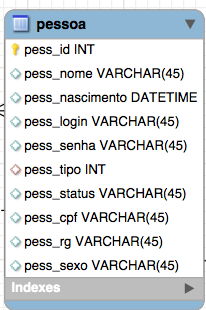
## *4.1.2.3 Tratamento de tabelas*

Assim como utilizado em *frameworks* conhecidos no mercado, como Hibernate e JPA, toda tabela do banco de dados deve possuir uma classe Java que seja sua cópia exata. Ou seja, uma classe Java, que tenha o mesmo nome da tabela, em que cada atributo da classe corresponda a nomenclatura e tipo de dados definidos na entidade de banco de dados.

A ferramenta reconhece que uma determinada classe Java representa uma tabela do banco de dados, desde que essa implemente uma interface de nome *TableMaster*. Uma interface *TableMaster* dará a uma determinada classe Java toda a estrutura necessária para a geração de códigos SQL dinâmicos. É graças a essa estrutura que o programador poderá optar por não escrever SQL no desenvolvimento, onde por consequência a ferramenta fará isso por ele.

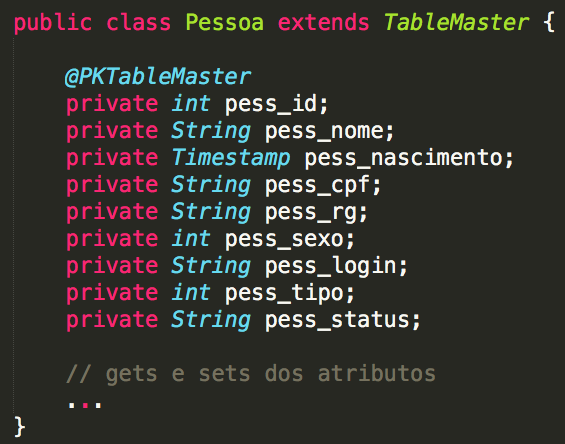
Tabelas no banco de dados podem ou não possuir chaves primárias. O fato da não existência de uma *primary key* numa tabela resulta em completa perda de eficiência do banco, assim como foge completamente da principal característica do modelo de entidade relacionamento, que são as *constraints*. Assim sendo, toda *TableMaster* possui uma a*nnotation* de nome @PKTableMaster, que deverá ser aplicada para algum atributo da classe que represente uma PK. A ferramenta identificará que o atributo que possuir essa *annotation* na definição da classe, será respectivo a chave primária da tabela. A figura 8 apresenta um exemplo de uma classe que representa uma tabela para a ferramenta, assim como a definição de uma chave primária nela.

**Figura 7 – Diagrama entidade relacionamento pessoa**



Fonte: Elaborada pelo autor.

**Figura 8 - Exemplo de definição de classe para tabela**



Fonte: Elaborado pelo autor.

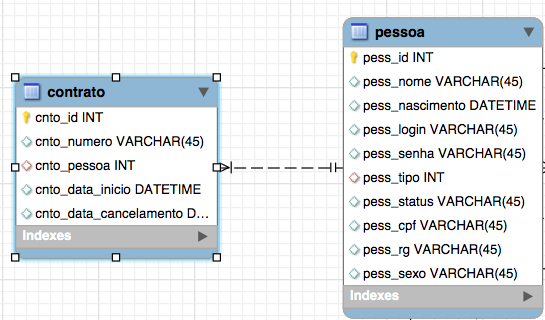
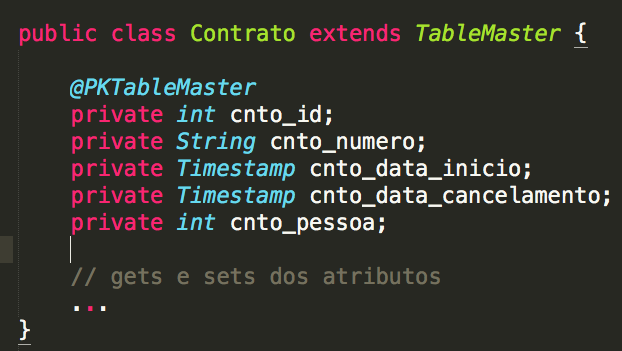
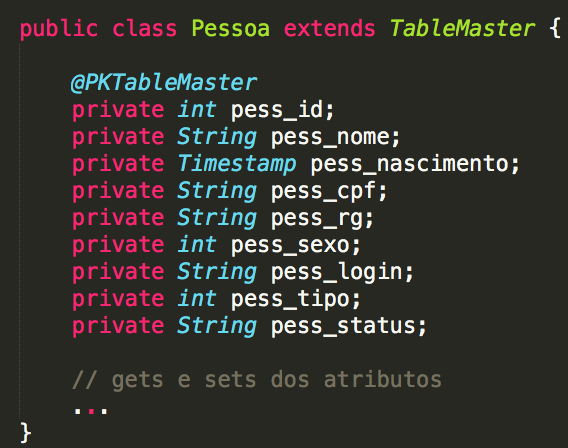
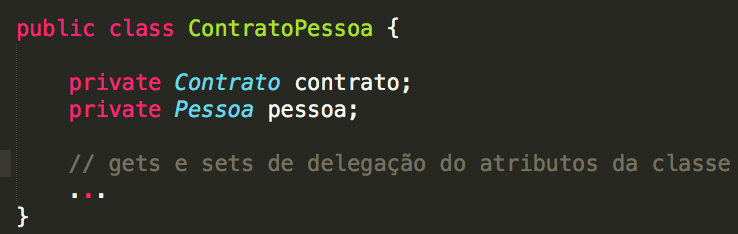
#### 4.1.2.4 Consultando e persistindo dados

Podemos observar por meio das figuras 7 e 8 que os métodos que realizam consultas e persistências da interface SigmaDB estão sempre esperando uma classe *TableMaster*. Ou seja, eles estarão sempre esperando uma classe que representa uma tabela do banco de dados.

Se um objeto *TableMaster* for informado ao método de consulta da classe SigmaDB, este, por sua vez, utilizará os valores das propriedades do objeto repassado como restrições para a consulta. Já se um objeto *TableMaster* for informado para o método de persistência, todos os valores contidos nas propriedades do objeto serão persistidos no banco, de forma que o atributo que representa a *primary key* irá ser a restrição para a persistência, tratando-se de *updates* ou *deletes*. O método de persistência exige que o usuário informe se ele precisa realizar um *INSERT*, *UPDATE* ou *DELETE* e essa informação é repassada por parâmetro na interface centralizadora do *framework*.

Seja o esquema do banco o apresentado na figura 9, vejamos um exemplo da realização de consultas e persistências pela ferramenta.

**Figura 9 - Diagrama entidade relacionamento com respectiva estrutura de classes**

Fonte: Elaborada pelo autor.

* **Consulta de dados:**

**Figura 10 - Estrutura de código de pesquisa de dados**



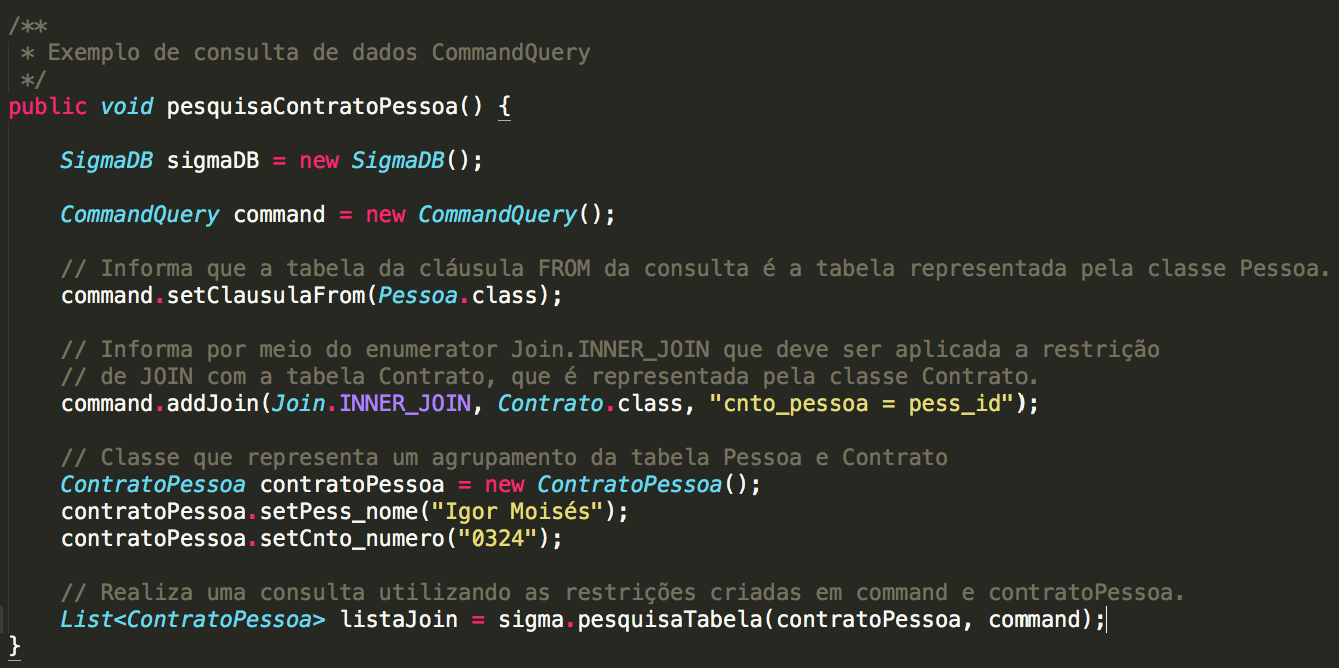
Fonte: Elaborada pelo autor.

Conforme ilustrado na figura 10, a interface centralizadora SigmaDB é capaz de realizar qualquer consulta a qualquer tabela do banco de dados. Para tanto basta informar instâncias das classes que devem ser consultadas para a ferramenta, e ela se responsabilizará pela consulta dos dados.

A ferramenta também permite a realização de consultas utilizando junções entre tabelas. Para tanto, conforme visto acima, o programador deverá criar uma classe que agrupe quais tabelas estarão aninhadas numa mesma consulta e repassar uma instância dessa classe para o método de consulta padrão.

O SigmaDB possui uma estrutura complementar chamada de *CommandQuery*. Essa estrutura serve como apoio a criação de restrições complementares a consulta. Por meio dessa é possível realizar a utilização de restrições dos tipo *IN*, ou *NOT IN*, em lista de elementos, utilização de *LIKE, BETWEEN*, *INNER JOIN*, *LEFT JOIN*, *IS NULL*, *IS NOT* *NULL*, *ORDER BY*, *GROUP BY*, etc. Essa estrutura é feita para o desenvolvimento de sintaxes SQL por meio da linguagem Java. Isto faz com que programadores que não tenham o devido conhecimento com a sintaxe SQL sinta-se mais capacitado para a escrever consultas de complexidade mais altas. Na figura 11 podemos ver a utilização dessa estrutura. A figura 12 apresenta suas principais funcionalidades.

**Figura 11 - Utilização de consultas com a CommandQuery**



Fonte: Elaborada pelo autor.

**Figura 12 - Principais funcionalidades da CommandQuery**

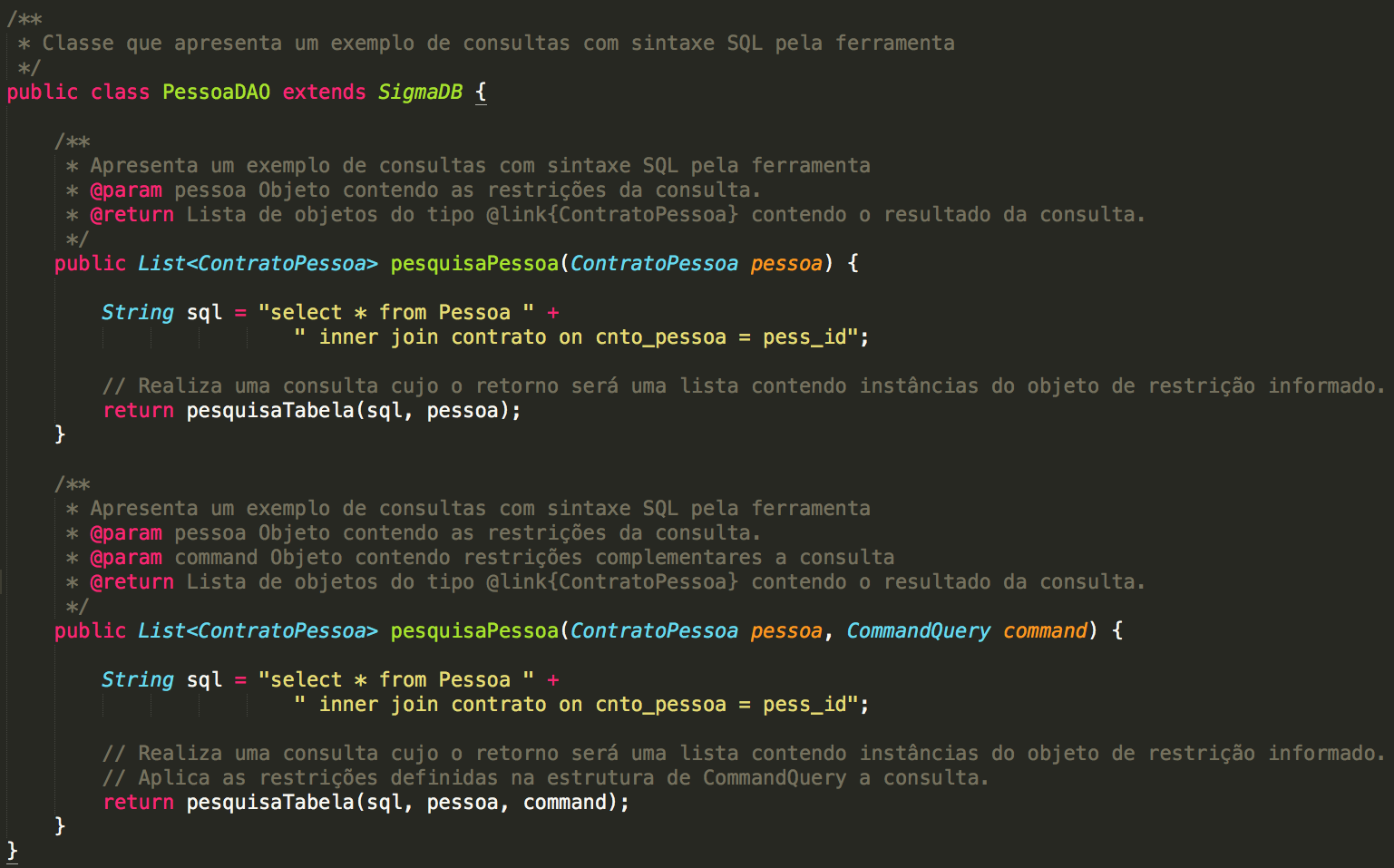


Fonte: Elaborado pelo autor.

O SigmaDB, além da funcionalidade da estrutura *CommandQuery*, ainda permite ao programador a escrita de sintaxes SQL. Com o pensamento de que haverá casos em que a escrita de sintaxe SQL seja necessária ao desenvolvimento, a classe SigmaDB possui métodos de acessibilidade do tipo *protected* que permitem que *Strings*, contendo sintaxes SQL, sejam informadas para consultas. Dessa forma, sempre que houver a necessidade de escritas SQL, será necessário a criação de uma nova classe que estenda as funcionalidades de SigmaDB, de forma que sintaxes SQL sejam inseridas e executadas por meio desta nova classe, que possui as funcionalidades específicas de SigmaDB.

Da mesma forma que nos outros métodos de consulta da ferramenta, a lista de objetos de retorno da consulta será sempre do mesmo tipo da instância informada como restrição. Conforme podemos observar na figura 14, o método de pesquisa (pesquisaPessoa) retornará uma lista de objetos do tipo ContratoPessoa. Isto deve-se ao fato de que no código podemos ver que foi informada uma instância de ContratoPessoa para o método pesquisaTabela. Caso o programador passasse uma instância de Pessoa ou uma instância de Contrato, o método pesquisaTabela iria retornar uma listagem contendo respectivas instâncias, que presentariam o resultado da consulta realizada.

**Figura 13 - Utilização de sintaxe SQL na ferramenta**



Fonte: Elaborada pelo autor.

* + **Persistência de dados:**

**Figura 14 - Estrutura de código de persistência de dados**



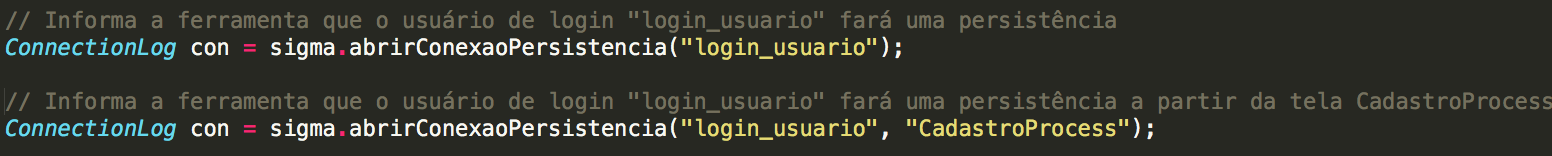
Fonte: Elaborada pelo autor.

Da mesma maneira que as consultas, persistências de dados no SigmaDB possuem um ponto único de requisição, conforme podemos observar na figura 15. A chamada ao método “applyUpdateTableMaster” é um ponto único de requisições a persistências de registros em tabelas. Essa estrutura, conforme podemos ver no comentário do código fonte, é capaz de realizar qualquer tipo de persistência, seja ela um *Insert, Update* ou *Delete*. Por meio do *enumerator* TypeOperation, o programador deverá escolher se ele deseja que a ferramenta realize um *Insert, Update* ou *Delete* no registro informado.

Ainda na figura 15 podemos ver que antes de realizar a persistência de dados, a ferramenta obriga ao programador que uma conexão com o banco seja aberta. Isso ocorre devido ao fato de que o programador terá a liberdade de manipular a conexão a sua vontade. A SigmaDB possui o método concluirConexao que realiza o *commit* de uma transição com o banco, assim como podemos ver na figura 7, a ferramenta também possui o método abortarConexao que realiza o *rollback* de todas as persistências realizadas na conexão aberta.

Ao persistir qualquer registro de alguma tabela do banco, a estrutura do SigmaDB gera automaticamente toda a estrutura de auditoria e versionamento de dados, se esse estiver habilitado no arquivo de configuração. Ao analisarmos a figura 15, podemos ver que sempre que algo é solicitado ao banco, é exigido uma conexão ativa. Essa conexão que ele solicita tem a função de servir de mapeamento para transação. Caso a opção de geração de *log* da ferramenta esteja ativa, a ferramenta obriga ao programador que ele informe um nome de usuário que está requisitando a persistência no objeto de conexão ativa com o banco. Conforme apresentado na figura 16, existe a possibilidade do programador informar, neste mesmo objeto, o local de onde a requisição partiu. Por exemplo, o nome de uma página ou tela. Caso ele não informe a ferramenta irá pegar o nome do método de onde partiu a chamada de persistência e irá inclui-lo no *log*.

**Figura 15 - Exemplo de abertura de conexões**



Fonte: Elaborada pelo autor.

## 4.1.3 O Ambiente do Sistema

Por se tratar de uma ferramenta para o auxílio no desenvolvimento de sistemas, o *framework* deverá ser utilizado por meio de IDEs de desenvolvimento para a tecnologia Java que proporcionem a utilização e importação de arquivos do tipo jar, que nada mais é do que um binário compactado de classes Java.

A ferramenta atuará nas estruturas de código definidas pelo programador, as quais atendam todos os requisitos explicados anteriormente nos itens 3.1.2.1, 3.1.2.2 e 3.1.2.3.

## 4.1.4 A definição do escopo

O objetivo deste projeto é apresentar uma ferramenta desenvolvida para a obtenção das características de agilidade, melhoria de estruturação de código e centralização de comandos relacionados a banco de dados.

A ferramenta deve ser flexível a qualquer ambiente de execução. Ou seja, deve ser independente do sistema operacional utilizado pela máquina em que ele estiver sendo executado. O sistema conta com as funcionalidades descritas abaixo:

* Realização de consultas: A ferramenta deve ser capaz de realizar qualquer consulta a qualquer tabela do sistema, de maneira que se um SQL não for informado para a consulta, a própria ferramenta deverá ser autônoma para gerar seu próprio SQL;
* Realização da persistência de dados: A ferramenta deve ser capaz, por meio de uma estrutura única, salvar registros em qualquer tabela do banco;
* Realização de auditoria e versionamento de dados: A ferramenta deverá ser capaz de registrar todo *commit* feito por ela no banco em forma de *log*, guardando data, usuário e o que foi alterado, inserido e/ou removido;

## 4.2 Motivação para o novo sistema

Conforme descrito anteriormente, a empresa utiliza-se de uma estrutura de classes de propriedade própria para a realização de conexão com seu banco de dados. Essa estrutura de classes causava certos transtornos devido a sua dificuldade de manutenção, necessidade de se conseguir códigos coesos e não redundantes, de maneira que o profissional da área de desenvolvimento de *software* conseguisse agilidade em seu trabalho.

A empresa tem em sua necessidade principal fazer com que seus programadores não percam tempo desenvolvendo instruções de banco para serem executadas. Seu interesse metodológico é voltado para que seus programadores foquem sua atenção para as regras de negócio de suas aplicações enquanto que a necessidades de tratamentos de transações sejam mantidas pela ferramenta.

## 4.3 Situação desejada

A ferramenta desenvolvida deverá atender as expectativas dos programadores que a utilizarão de maneira que esses, por intermédio dela, consigam construir códigos menos complexos, mais coesos, acoplados e principalmente que ganhem uma grande velocidade de trabalho não perdendo mais tempo com desenvolvimento de estruturas de trabalho para SQL.

## 5 O SISTEMA PROPOSTO

“Todo e qualquer sistema deve ser modelado antes de se iniciar a sua implementação, entre outras coisas, porque os sistemas de informação frequentemente costumam possuir a propriedade de ‘crescer’” (GUEDES, 2008, p.19). A análise do software antes de sua implementação se faz necessária para a definição de todo o escopo de trabalho, fazendo com que as necessidades do usuário sejam devidamente atendidas.

A modelagem do SigmaDB se deu a garantir o suprimento de todas as deficiências apontadas pelo seu cliente no momento do trabalho de codificação. Dessa forma, se fez necessário realizar a documentação de todos os requisitos para que, a partir daí, a solução SigmaDB possa ser aplicada.

## 5.1 Lista de Requisitos do Sistema

Dividida entre requisitos funcionais e requisitos não funcionais, a análise de requisitos é a primeira parte fase de desenvolvimento de *softwares.* É por meio desta que as necessidades funcionais e não funcionais dos clientes são listadas, analisadas e estudadas.

Num projeto de *software* esta é a fase em que as ideias são colocadas na mesa e toda a compreensão das necessidades são compartilhadas e compreendidas entre alguns integrantes do projeto.

## 5.1.1 Requisitos Funcionais

Os requisitos funcionais definem como o sistema se comporta, através das declarações das funções que o sistema deve oferecer, de como o sistema deve reagir a entradas específicas e de como deve se comportar em determinadas situações, ou ainda define restrições explícitas do que o sistema não deve fazer (SOMMERVILLE, 2003).

**Quadro 1 - Elementos Funcionais**

|  |  |
| --- | --- |
| **Código** | **Descrição** |
| RF\_01 | **Gerenciamento de conexão com bancos:**  A ferramenta deve ser capaz de realizar conexão com qualquer banco de dados do tipo relacional.  Para tanto, deverá dispor de uma estrutura genérica que seja capaz de ser configurada externamente no projeto que irá encapsular toda a API do *Framework*.  No projeto externo, deverá ser criado, dentro da pasta *source*, um arquivo de nome *SigmaDBProperties*.xml.  Neste arquivo deverá ser incluído as *tags* abaixo:   * *userName* (Nome do usuário do banco de dados); * *password* (Senha do usuário do banco de dados); * URL (URL de conexão com o banco de dados); * *driver* (Nome do driver de conexão com o banco de dados); * *printSql* (Boleano que indica se qualquer estrutura sql montada pela ferramenta, deva ser escrita no *console* da aplicação); * *iLog* (Boleano que indica se a estrutura de log estará ativada ou desativada) |
| RF\_02 | **Mapeamento de tabelas:**  A ferramenta deverá ser capaz de mapear classes e tabelas.  Para tanto deverá ser desenvolvida uma interface que indique que uma determinada classe represente uma tabela do banco de dados.  Essa interface deverá ser chamada de *TableMaster* e será responsável por realizar o mapeamento de tudo o que vem a ser uma tabela para a ferramenta. Essa estrutura também deverá possuir uma *annotation* capaz de identificar qual atributo da classe representará uma *primary key*. |
| RF\_03 | **Gerenciamento de Logs:**  Toda tupla inserida no banco, independentemente de tabela, deverá ser auditada por meio de uma estrutura que armazene:  data/hora do ocorrido, identificador de quem requisitou a ação, local de onde partiu a requisição, uma numeração única para a transação que será chamada de versão.   * Identificador de registro (Identificador único para o registro de log); * Data/hora (Data e hora em que ocorreu o evento de persistência de um dado); * Identificador de usuário (Alguma informação que identifique um usuário que realizou a persistência); * Local (Local de onde partiu a requisição da persistência); * Identificador de transação (Identificador único para a transação) * Conteúdo existente antes da alteração |
| RF\_04 | **Gerenciamento da estrutura de consultas sem sql:**  Para utilizar as estruturas de pesquisa, a ferramenta deverá disponibilizar interfaces para que se possa realizar as devidas ações.  Para tanto deverá ser disponibilizada uma interface que seja capaz de receber qualquer instância de classe e realizar uma consulta a base de dados.  A ferramenta deverá ser capaz de identificar se a instância representa uma ou mais tabelas e desta forma realizar a respectiva consulta, tomando como restrições da consulta todos os valores das propriedades da instância informada.  O retorno da consulta deverá ser uma lista de instâncias, do mesmo tipo da informada a ferramenta, contendo o resultado da consulta. |
| RF\_05 | **Gerenciamento da estrutura de consultas com sql:**  Para utilizar as estruturas de pesquisa, a ferramenta deverá disponibilizar interfaces para que se possa realizar as devidas ações.  Para tanto deverá ser disponibilizada uma interface que seja capaz de receber qualquer instância de classe, uma consulta no formato sql e realizar uma consulta a base de dados, tomando como base o sql informado pelo usuário.  O retorno da consulta deverá ser uma lista de instâncias, do mesmo tipo da informada a ferramenta, contendo o resultado da consulta. |
| RF\_06 | **Gerenciamento da estrutura de persistência:**  Para utilizar as estruturas de persistência de dados, a ferramenta deverá disponibilizar interfaces para que se possa realizar as devidas ações.  Para tanto, deverá ser disponibilizada uma interface que se permita realizar qualquer tipo de persistência de dados. Será um único ponto de acesso onde se possa realizar as ações de *Inserts, Updates* e *Deletes*.  Esta estrutura deverá receber uma instância de classe que represente uma tabela do banco de dados, com seu respectivo tipo de ação solicitado, seja um *Insert, Update* ou *Delete*.  Baseado nos valores das propriedades da instância informada, a ferramenta deverá ser capaz de realizar a ação solicitada.  Como retorno da ação, a ferramenta deverá sempre retornar o id (*primary key*) do registro trabalhado, se houver. |

Fonte: Elaborado pelo autor.

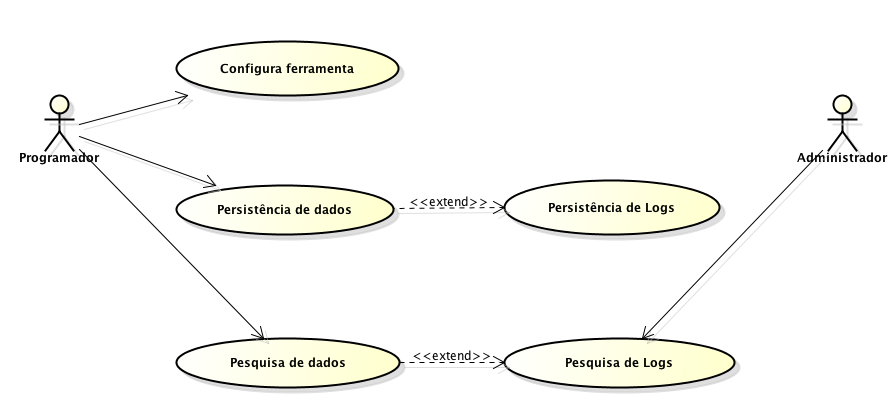
## 5.1.2 Requisitos Não Funcionais

**Quadro 2 - Requisitos Não Funcionais**

|  |  |
| --- | --- |
| Código | Descrição |
| RNF\_01 | **Estrutura complementar de consultas**  A ferramenta deverá disponibilizar para o usuário uma estrutura auxiliar para o desenvolvimento de consultas. Essa estrutura deverá ser chamada de *CommandQuery* e será uma interface capaz de realizar as ações descritas abaixo:   * Permitir realizar restrições do tipo *IN* e *Not IN* em lista de objetos; * Permitir realizar restrições dentro do período de datas e tempos; * Permitir realizar restrições fora do período de datas e tempos; * Permitir realizar verificações elementos nulos em atributos de tuplas, tais como os comandos sql *isnull* e *is not null*; * Permitir realizar *joins* com qualquer tabela, fazendo com que consultas que apontem para uma única tabela possa realizar *joins* com qualquer tabela baseado na vontade e necessidade do usuário; * Permitir incluir novas restrições a consulta, além dos valores informados nas propriedades da instância de objetos informados para a interface de captação de consultas da ferramenta. Estas restrições podem ser cláusula AND e cláusulas *OR*. |
| RNF\_02 | **Desenvolvimento de consultas:**  Baseado em RNF\_01, a ferramenta deverá disponibilizar uma estrutura que seja capaz de realizar o desenvolvimento de qualquer consulta por meio de código Java, sem que seja necessário o usuário escrever nenhum comando SQL.  Para tanto, na estrutura de RNF\_01, deverá ser criado uma maneira para que seja inserida um nome de tabela e através da estrutura de *joins* e o restante das restrições que estarão preparadas, possa-se realizar qualquer tipo de consulta, onde o retorno desta será o mesmo retorno descritos em RNF\_04. |

Fonte: Elaborado pelo autor.

## 5.2 Diagrama de Caso de Uso



**Figura 16 - Diagrama de caso de uso**

Fonte: Elaborado pelo autor.

## 5.3 Especificações dos casos de uso

Segundo Guedes (2008), casos de uso declaram serviços tarefas ou funções que podem ser utilizados de alguma forma pelos usuários do sistema. Servem para exemplificar e documentar comportamentos e funções do sistema.

## 5.3.1 Configura Ferramenta

O caso de uso Configura Ferramenta possibilita que o usuário programador configure a ferramenta dentro do seu projeto de desenvolvimento.

* **Atores:** Programador
* **Pré-condições:** O usuário programador deverá possuir instalado em seu computador uma IDE de desenvolvimento da tecnologia Java e o arquivo SigmaDB.jar, respectivo ao *Framework*.
* **Pós-condições:** Possuir a ferramenta pronta para o auxílio do desenvolvimento da aplicação.
* **Fluxo Principal:**

**Quadro 3 - Quadro de Fluxo Principal**

|  |  |
| --- | --- |
| **Passo** | **Descrição** |
| 1 | O fluxo principal inicia quando o usuário programador cria um novo projeto e inclui a biblioteca SigmaDB.jar no seu *classpath*. |
| 2 | O programador deverá criar no seu projeto, dentro da pasta *source*, um arquivo de nome SigmaDoProperties.xml.  Neste arquivo deverá haver os atributos descritos abaixo, preenchidos com seus respectivos valores.   * *userName* (Nome do usuário do banco de dados); * *password* (Senha do usuário do banco de dados); * url (Url de conexão com o banco de dados); * *driver* (Nome do driver de conexão com o banco de dados); * *printSql* (Boleano que indica se qualquer estrutura sql montada pela ferramenta, deva ser escrita no console da aplicação); * *iLog* (Boleano que indica se a estrutura de auditoria deverá estar ativada ou desativada) |

Fonte: Elaborado pelo autor.

## 5.3.2 Persistência de dados

O caso de uso Persistência de dados possibilita que o usuário programador configure a ferramenta para que ela consiga persistir informações na base de dados.

* **Atores:** Programador
* **Pré-condições:** O usuário programador deverá ter sua base de dados já formulada e implementada.
* **Pós-condições:** Dados persistidos na base de dados
* **Fluxo Principal:**

**Quadro 4 - Quadro de Fluxo Principal**

|  |  |
| --- | --- |
| **Passo** | **Descrição** |
| 1 | O Fluxo principal inicia com o usuário programador definindo classes Java que representem cada entidade do seu banco de dados. Cada classe que representa uma entidade do banco de dados deve possuir atributos que sejam respectivos aos nomes de cada atributo da entidade do banco de dados, assim como seu tipo. |
| 2 | O usuário programador deverá fazer com que cada classe que represente uma entidade do seu banco de dados, estenda a classe de nome *TableMaster* que estará dentro da biblioteca SigmaDB, incluída no *classpath* do projeto. |
| 3 | Se a entidade do banco de dados possuir uma *Primary Key*, o atributo respectivo da classe Java deverá receber a *annotation* @PkTableMaster. |
| 4 | O usuário programador deverá abrir uma conexão com o banco chamando o método abrirConexaoPersistencia, que retorna uma instância da classe Connection, da classe SigmaDB. |
| 5 | De posse de uma conexão aberta o usuário poderá repassar qualquer *TableMaster* para o método *applyUpdateTableMaster* da classe SigmaDB, repassando também a conexão que ele possui aberta e qual o tipo de persistência ele deseja fazer. Ao realizar a chamada ao método *applyUpdateTableMaster*, o programador deve informar também qual o tipo de persistência ele necessitará, através de um *enumerator TypeOperation*. Este *enumerator* listará o *Insert, Update* ou *Delete*. |
| 6 | O usuário programador, após o término de todas as suas persistências, chamará o método concluirConexao, da classe SigmaDB, informando como parâmetro a conexão aberta, ou a conexão que ele deseja que seja comitada na base de dados. |

Fonte: Elaborado pelo autor.

## 5.3.3 Pesquisa de dados

O caso de uso Pesquisa de dados possibilita que o usuário programador configure a ferramenta para que ela consiga consultar informações na base de dados.

* **Atores:** Programador
* **Pré-condições:** O usuário programador deverá ter sua base de dados já formulada e implementada. O usuário precisa ter implementado em classes Java cópias idênticas de suas entidades do banco, tanto em nome, quanto em nomes de atributos com seus respectivos tipos.
* **Pós-condições:** Lista de dados contendo o resultado das consultas.
* **Fluxo Principal:**

**Quadro 5 - Quadro de Fluxo Principal**

|  |  |
| --- | --- |
| **Passo** | **Descrição** |
| 1 | O fluxo principal inicia-se com o usuário programador criando uma instância de um *TableMaster* que represente alguma entidade do banco de dados. |
| 2 | O usuário programador deverá pegar a instância criada e preencher os atributos, de maneira que cada atributo será considerado como uma restrição para a consulta que será realizada. |
| 3 | Com os devidos atributos já preenchidos, o usuário programador chamará o método pesquisa da classe SigmaDB, passando como parâmetro a instância de TableMaster criada. |
| 4 | Como retorno, o usuário programador, receberá uma lista de instâncias do tipo que foi enviada para a consulta contendo o resultado da consulta. |

Fonte: Elaborado pelo autor.

* **Fluxos Alternativos:**

**Quadro 6 - Fluxo Alternativo - Consulta com SQL**

|  |  |
| --- | --- |
| **Consulta com SQL** | |
| **Passo** | **Descrição** |
| 1 | O usuário programador deverá criar uma nova classe que estenda a classe SigmaDB. |
| 2 | O usuário programador deverá montar métodos de pesquisa que recebam como parâmetro as restrições da consulta. |
| 3 | Neste método o programador deverá escrever uma String contendo uma sintaxe SQL de consulta. Com a estrutura da pesquisa SQL montada, o programador deverá informar uma instância dos objetos que serão retornados pela consulta, e através do método pesquisa de acessibilidade protected da classe SigmaDB é possível realizar uma consulta informando a instância de restrições e o SQL. |
| 4 | Como retorno, o usuário programador, receberá uma lista de instâncias do tipo que foi enviada para a consulta contendo o resultado da consulta. |

Fonte: Elaborado pelo autor.

**Quadro 7 - Fluxo Alternativo - Consultas Dinâmicas**

|  |  |
| --- | --- |
| **Consultas Dinâmicas** | |
| **Passo** | **Descrição** |
| 1 | O usuário programador deverá criar uma instância de uma estrutura auxiliar chamada de *CommandQuery*. O *CommandQuery* é uma classe que servirá para desenhar consultas dinâmicas. |
| 2 | Com a instância de *CommandQuery*, o usuário poderá preencher a propriedade “*from*” do objeto, assim como as propriedades “*join”*. Ambas as propriedades recebem como parâmetro Objetos do tipo *Class*, que devem obrigatoriamente representar um *TableMaster*, ou seja um espelho da entidade do banco de dados. |
| 3 | O usuário Programador poderá definir todas as restrições da consulta no objeto *CommandQuery*, que conta com uma estrutura de todos os tipos de restrições sql para consulta. |
| 4 | O método pesquisa da classe SigmaDB está sobrecarregado com uma chamada que aceita como parâmetro uma instância de *CommandQuery*.  Uma instância dos objetos que deverão ser retornados como resultado da consulta e uma instância de *CommandQuery* deverão ser repassados ao método “pesquisa”. |
| 5 | Como retorno, o usuário programador, receberá uma lista de instâncias do tipo que foi enviada para a consulta contendo o resultado da consulta. |

Fonte: Elaborado pelo autor.

## 5.3.4 Persistência de *logs*

O caso de uso Persistência de *logs* possibilita que o usuário programador configure a ferramenta para que ela consiga persistir informações de auditoria de dados na base de dados.

* **Atores:** Programador
* **Pré-condições:** O usuário programador deverá ter sua base de dados já formulada e implementada. O usuário precisa ter implementado em classes Java cópias idênticas de suas entidades do banco, tanto em nome, quanto em nomes de atributos com seus respectivos tipos.
* **Pós-condições:** *Logs* persistidos na base de dados
* **Fluxo Principal:**

**Quadro 8 - Fluxo Principal**

|  |  |
| --- | --- |
| **Passo** | **Descrição** |
| 1 | O fluxo principal inicia-se com o usuário programador criando uma nova entidade no banco de dados. Esta entidade deverá chamar-se de “log” e deverá conter os atributos abaixo.   * + *Id* : *Integer*   + versao: *Integer*   + tipo: *Varchar*   + tabela – *Varchar*   + pk\_tabela – *Integer*   + data\_hora – *Timestamp*   + origem – *Varchar*   + usuario – *Integer*   + valores - *Varchar* |
| 2 | Quando o usuário programador realizar qualquer persistência de dados, a ferramenta se responsabiliza de persistir as informações da entidade *Log* respectivas ao registro que está sendo persistido. |

Fonte: Elaborado pelo autor.

## 5.3.5 Pesquisa de *logs*

O caso de uso Pesquisa de *logs* possibilita que o usuário programador configure a ferramenta para que ela consiga consultar informações de auditoria de dados na base de dados.

* **Atores:** Programador, Administrador
* **Pré-condições:** O usuário programador deverá ter sua base de dados já formulada e implementada. O usuário precisa ter implementado em classes Java cópias idênticas de suas entidades do banco, tanto em nome, quanto em nomes de atributos com seus respectivos tipos. O usuário precisa possuir a entidade *Log* implementada.
* **Pós-condições:** Lista de dados contendo o resultado das consultas.
* **Fluxo Principal:**

**Quadro 9 - Fluxo Principal**

|  |  |
| --- | --- |
| **Passo** | **Descrição** |
| 1 | O fluxo principal inicia-se com o usuário programador criando uma instância da classe *Log*. |
| 2 | Com a instância criada o usuário programador deverá preencher os atributos da instância, de maneira que cada atributo preenchido seja respectivo a uma restrição da consulta. |
| 3 | Feito o preenchimento dos atributos que servirão como restrições da consulta, o usuário deverá chamar o método de pesquisa da classe SigmaDB, passando como parâmetro a instância da entidade *Log*. |
| 4 | Como resultado, o usuário programador receberá uma lista de instâncias da classe *Log*, contendo o resultado da consulta. |

Fonte: Elaborado pelo autor.

* **Fluxo Alternativo:**

**Quadro 10 - Fluxo Alternativo - Utilização do Administrador**

|  |  |
| --- | --- |
| Utilização do Administrador | |
| **Passo** | **Descrição** |
| 1 | O fluxo alternativo inicia com o usuário Administrador acessando a base de dados. |
| 2 | Ao realizar acesso a entidade *Log* pela base de dados, o usuário administrador consegue ter a mesma visão do usuário Programador, consultando a estrutura de *log* da maneira como necessitar. |
| 3 | Como saída o usuário administrador obtém o resultado da consulta que lhe for necessária. |

Fonte: Elaborado pelo autor.

## 5.4 Modelo Conceitual de Classes

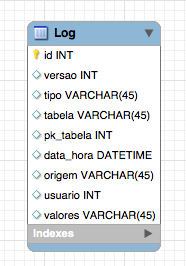


**Figura 17 - Modelo Conceitual de Classes**

Fonte: Elaborado pelo autor.

## 5.5 Modelo Conceitual de Dados

**Figura 18 - Modelo conceitual de dados**

****

Fonte: Elaborado pelo autor.

## 6 CONCLUSÕES

Com o crescimento do mercado de desenvolvimento ágil de *software*, seguido por suas exigências, faz-se necessário a criação de um modelo de negócio que seja rápido, intuitivo e de fácil manuseio. A partir desta premissa, é que identificamos a necessidade do investimento nessa ferramenta, de maneira que um programador possa, a partir de sua utilização, trabalhar de maneira mais ágil.

Entretanto, sabemos que para elaborar e implementar qualquer tipo de sistema, faz-se necessário a realização de um levantamento das necessidades dos usuários, de maneira que se possa identificar quais são os pontos mais críticos para o desenvolvimento do trabalho deste. Este estudo tomou como base os pontos de maior tempo gasto em desenvolvimento, dentro da empresa Cientz, tendo como intenção trabalhar em cima de melhorias que objetivem trazer a redução desse período de trabalho.

Com a intenção de ser um diferencial dentre as ferramentas do gênero propostas atualmente no mercado, a aplicação desenvolvida neste estudo traz como novidades a possibilidade da escrita de consultas sql, assim como o versionamento automático de persistências em tabelas.

Diante das análises e levantamentos das funcionalidades, identificamos a necessidade da criação e manutenção de artefatos para o projeto: casos de uso, elicitação e documentação de requisitos funcionais e não funcionais.

Diante do exposto, espera-se que a ferramenta possa oferecer aos seus usuários um serviço de desenvolvimento ágil, robusto e inovador.

Concluímos, desta maneira, que a implantação da ferramenta nos projetos de desenvolvimento trará um conjunto de benefícios ao modelo de negócio, independentemente de qual escopo ele esteja incluído, tendo em vista a agilidade de desenvolvimento obtida para o trabalho de programação, assim como a segurança dos dados.

# REFERÊNCIAS

ALEXANDER, C.; ISHIKAWA, S.; SILVERSTEIN, M. **A pattern language**. Oxford: Oxford University Press, 1977.

ANÁLISE de Requerimento de Software. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2017. Disponível em: <https://pt.wikipedia.org/w/index.php?title=An%C3%A1lise\_de\_requerimento\_de\_software&oldid=48545612 >. Acesso em: 14 abr. 2017.

CODD, E. F. A relational model of data for large shared data banks. Communications of the ACM, v. 13, n. 6, p. 377-387, 1970. Disponível em: <http://dl.acm.org/citation.cfm?id=362685>. Acesso em: 20 maio 2017.

DEITEL, J; DEITEL, M. **Java Como Programar**. 8. ed. São Paulo: Pearson, 2010.

EDGAR Frank Codd. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2014. Disponível em: <<https://pt.wikipedia.org/w/index.php?title=Edgar_Frank_Codd&oldid=38665374>>. Acesso em: 10 jan. 2017.

ELMASRI, R.; NAVATHE, S. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson, 2011.

FREEMAN, E.; FREEMAN, E. **Use a cabeça**: padrões de projetos. 2. ed. Jacaré: Alta Books, 2007.

GUEDES, A. **UML**: uma abordagem prática. 3. ed. São Paulo: Novatec, 2008.

JDBC. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2016. Disponível em: <<https://pt.wikipedia.org/w/index.php?title=JDBC&oldid=46366808>>. Acesso em: 10 jan. 2017.

KOCHE, J. C. **Fundamentos de metodologia científica**: teoria da ciência e iniciação à pesquisa. 28. ed. Petrópolis: Vozes, 2006.

KOTLER, P. **Administração de marketing**: análise, planejamento, implementação e controle. 5. ed. São Paulo: Atlas, 1998.

MÁQUINA Virtual Java. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2017. Disponível em: <<https://pt.wikipedia.org/w/index.php?title=M%C3%A1quina_virtual_Java&oldid=47985684>>. Acesso em: 09 abr. 2017.

MÁQUINA Virtual. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2016. Disponível em: <<https://pt.wikipedia.org/w/index.php?title=M%C3%A1quina_virtual&oldid=47421142>>. Acesso em: 9 abr. 2017.

MARTIN, R. **Código Limpo**. [S.I.]: Alta Books, 2008.

ODBC. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2017. Disponível em: <<https://pt.wikipedia.org/w/index.php?title=ODBC&oldid=47827286>>. Acesso em: 10 jan. 2017.

POPEK, G.; GOLDBERG, R. Formal requirements for virtualizable 3rd generation architectures. **Communications of the ACM**, v.17, n.7, pp. 412-421, 1974.

SILBERSCHATZ A.; KORTH, H.; SUDARSHAN, S. **Sistemas de Bancos de Dados**. 5. ed. Rio de Janeiro: Campos, 2006.

SOMMERVILLE, I. **Engenharia de Software**. Glasgow: Addison-Wesley, 2003.

SQL. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2016. Disponível em: <<https://pt.wikipedia.org/w/index.php?title=SQL&oldid=47157351>>. Acesso em: 1 fev. 2017.

WEINBERGER, M. **The 9 most popular computer languages, according to the Facebook for programmers**. Disponível em: <http://www.businessinsider.com/github-most-popular-coding-languages-2015-8>. Acesso em: 9 abr. 2017.