PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA POLITÉCNICA E DE ARTES

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

BUSINESS INTELLIGENCE E GESTÃO DE NEGÓCIO: ESTUDO DE CASO SOBRE A APLICAÇÃO DE BUSINESS INTELLIGENCE NA OBTENÇÃO DE INFORMAÇÕES

FELIPE ANDRADE DE PAIVA

GOIÂNIA,

2023

FELIPE ANDRADE DE PAIVA

BUSINESS INTELLIGENCE E GESTÃO DE NEGÓCIO: ESTUDO DE CASO SOBRE A APLICAÇÃO DE BUSINESS INTELLIGENCE NA OBTENÇÃO DE INFORMAÇÕES

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Me. Gustavo Vinhal.

GOIÂNIA,

2023

FELIPE ANDRADE DE PAIVA

BUSINESS INTELLIGENCE E GESTÃO DE NEGÓCIO: ESTUDO DE CASO SOBRE A APLICAÇÃO DE BUSINESS INTELLIGENCE NA OBTENÇÃO DE CONHECIMENTO

Este Trabalho de Conclusão de Curso julgado adequado para obtenção o título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, em ____/____/_____.

_____

Profa. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca Examinadora:

_____

Orientador: Prof. Msc. Gustavo Siqueira Vinhal

_____

Prof. Dr. Rafael Leal Martins

_____

Profa. Msc. Ana Flávia Marinho de Lima Garrote

GOIÂNIA

2023

RESUMO

Este trabalho implementa um sistema de *Business Intelligence* (BI) em todas as suas etapas, o objetivo deste estudo é melhorar a gestão de processos e a tomada de decisão em empresas, especificamente em relação ao setor de telecomunicações da empresa. O foco do desenvolvimento do sistema está em melhorar a gestão de telecomunicações em empresas, por se tratar de um setor ainda muito importante para o funcionamento de uma empresa, tão como para seu crescimento.

Sendo assim, o sistema de *BI* desenvolvido tem como fonte de dados para a análise um sistema de comunicação unificado e baseado na *WEB*, esse sistema de comunicação se comporta como uma central que recebe e realiza chamadas, registra informações sobre essas chamadas, fornecendo funções de gerenciamento e muito mais. Esse sistema é utilizado pela empresa utilizada como alvo dos testes, e o objetivo da implementação do sistema de *BI* é mostrar como o gerenciamento é impactado positivamente quando os dados são tratados e disponibilizados para análise da forma correta.

No desenvolvimento do estudo, foi realizada uma pesquisa e uma revisão sistemática da literatura com o objetivo de esclarecer a teoria acerca do desenvolvimento de um sistema de *BI*. O sistema foi desenvolvido utilizando técnicas de modelagem de banco de dados, para a criação do banco que irá receber os dados extraídos, projeto de algoritmos e técnicas de programação, para a criação de códigos que irão auxiliar e tornar mais ágil o processo de execução do sistema, além do uso de ferramentas para o auxílio da análise e visualização dos resultados.

Palavras chave: *Business Intelligence*. *Data Warehouse*. Gerenciamento. Análise de dados. Extração, transformação e carga de dados.

ABSTRACT

This work implements a Business Intelligence (BI) system in all its stages, aiming to enhance process management and decision-making within companies, specifically in the telecommunications sector of the company. The focus of the system's development is to improve telecommunications management in companies, as it remains a crucial sector for a company's operation and growth.

Therefore, the BI system developed uses a unified, web-based communication system as its data source for analysis. This communication system acts as a hub that receives and makes calls, records information about these calls, provides management functions, and more. The system is utilized by the company chosen for testing, and the goal of implementing the BI system is to demonstrate how management is positively impacted when data is processed and made available for analysis in the correct manner.

In the course of the study, a systematic literature review and research were conducted to clarify the theory related to BI system development. The system was developed using database modeling techniques for creating the database to receive the extracted data, algorithm design, and programming techniques for creating code to assist and streamline the system execution process, as well as the use of tools for analysis and visualization of the results.

Keywords: Business Intelligence. Data Warehouse. Management. Data analysis. Extract, Transform, Load of data.

LISTA DE FIGURAS

LISTA DE SIGLAS

TIC	Tecnologia da Informação e Comunicação
BI	*Business Intelligence*
DW	*Data Warehouse*
ETL	*Extract, Transform and Load*
OLAP	*Online Analytical Processing*
MOLAP	*Multidimensional Online Analytical Processing*
ROLAP	*Relational Online Analytical Processing*
DOLAP	*Desktop Online Analytical Processing*
HOLAP	*Hybrid Online Analytical Processing*

SUMÁRIO

1. Introdução

É notório que os avanços e o uso massivo das Tecnologias da Informação e Comunicação (TIC) influenciam o comportamento de um coletivo social. Tal fato instiga amplos debates no campo da Ciência da Informação no que tange, principalmente, a utilização salutar de dados, informação e conhecimento gerados a partir dos artefatos computacionais (câmeras, celulares, cartões de crédito, sensores de vários tipos, etc.). Por isso, admite-se a necessidade da discussão interdisciplinar sobre a utilização desses dados para a obtenção de conhecimento, relacionando-os com o fenômeno tecnológico denominado *Big Data* (EIICA, 2019).

O *Big Data* é um termo derivado dos avanços recentes relativos à massificação da utilização de recursos tecnológicos e da farta produção de dados. Em suma, é um conceito que caracteriza volumosos conjuntos de dados heterogêneos. (RAUNTENBERG et al., 2019)

Corroborando essa visão de mundo, mediante o avanço da Internet, a humanidade vem produzindo cada vez mais dados nas mais variadas plataformas digitais. Bugnion, Manivannan e Nicolas (2017) pontuam que cerca de 90% dos dados produzidos são resultado do uso intenso das Tecnologias de Informação e Comunicação nos últimos tempos. Por conseguinte, os dados são abundantemente e velozmente produzidos, servindo de matéria-prima para tomada de decisão em grandes corporações (ECONOMIST, 2017).

Com o advento tecnológico, cada vez são produzidas bases de dados maiores e mais precisas para variados domínios. Neste contexto, desenvolver ferramentas voltadas para utilizar esses dados como uma forma de extrair informações e conhecimentos para auxiliar no processo de gestão de negócio tem se tornado cada vez mais comum no mundo dos negócios. Devido a isso, o desenvolvimento de soluções que utilizam técnicas de coleta, organização, análise e compartilhamento para obterem conhecimento das enormes bases de dados é foco de investimento por grandes organizações (RAUNTENBERG et al., 2019).

Business Intelligence (BI) emergiu como uma importante área de estudo para ambos os profissionais e pesquisadores, refletindo a magnitude e o impacto dos

problemas relacionados a dados a serem resolvidos na contemporaneidade das organizações. (MIS Quartely, 2012, p. 1165)

Business Intelligence é um termo cunhado por Howard Dresner do Gartner Group, em 1989, para descrever um conjunto de conceitos e métodos para melhorar o processo de tomada de decisão das empresas, utilizando-se de sistemas fundamentados em fatos e dimensões. O *BI* baseia-se em agrupar informações de diversas fontes e apresentá-las de forma unificada e sob uma métrica comum. (BRAGHITTONI, 2017)

Segundo Howard Dresner, o *BI* é uma metodologia pela qual se estabelecem ferramentas para obter, organizar, analisar e prover acesso às informações necessárias aos tomadores de decisão das empresas para analisarem os fenômenos acerca de seus negócios. (DRESNER, 1989)

Uma grande quantidade de dados é acumulada dentro dos sistemas de informação de empresas públicas e privadas. Esses dados são originários de transações internas e de fontes externas. No entanto, mesmo que tenham sido coletados e armazenados de forma sistemática e estruturada, esses dados não podem ser utilizados diretamente para fins de tomada de decisão. Eles precisam ser processados por meio de ferramentas de extração apropriadas e métodos analíticos capazes de transformá-los em informações e conhecimentos que podem ser posteriormente usados pelos tomadores de decisão. (VERCELLIS, 2011)

Várias empresas e organizações públicas e privadas desenvolveram nos últimos anos mecanismos formais e sistemáticos para colher, armazenar e compartilhar seu conhecimento, visto agora como um ativo intangível e inestimável. É evidente que a gestão de conhecimento e a *Business Intelligence* compartilham algum grau de familiaridade em seus objetivos. O principal objetivo de ambas as disciplinas é desenvolver ambientes que possam apoiar os profissionais que detém o conhecimento nos processos de tomada de decisão e atividades complexas de resolução de problemas. (VERCELLIS, 2011)

As telecomunicações nas empresas são importantes para garantir o bom funcionamento dos canais de comunicação. Dentre os canais que fazem parte do sistema de telecomunicações, a telefonia fixa e móvel são essenciais para qualquer negócio hoje, por meio disso, acontece grande parte da interação entre empresa e

cliente. Por meio dos dados massivos obtidos pela telefonia, é possível uma empresa analisar suas estratégias de comunicação, a organização e a qualidade do seu departamento telefônico, visando assim a melhoria do atendimento, um melhor posicionamento do seu negócio no mercado, além de atrair mais clientes e aumentar sua chance de fidelização. (HENRIQUE, 2021)

Diante do contexto, esse projeto visa responder a seguinte questão de pesquisa: - **Como aplicar conceitos de Business Intelligence para otimizar a gestão de processos em telecomunicações empresarial?**

Este trabalho tem o objetivo geral de demonstrar por meio da aplicação de algoritmos e ferramentas de análise e visualização de dados a implementação de um sistema de *BI* para auxiliar o gerenciamento da telecomunicação em uma empresa.

Os objetivos específicos:

a) Apresentar as técnicas e teorias que envolvem um ciclo de *BI*.

b) Avaliar os casos de uso da aplicação de *BI* para a obtenção de relatórios informativos sobre o funcionamento da telecomunicação empresarial.

c) Demonstrar a importância da aplicação de *BI* nas tomadas de decisão.

Espera se que o resultado desse trabalho possa contribuir:

a) Na aplicação real dos conceitos de *BI* para obter conhecimento e informações que auxiliem no gerenciamento.

b) Valorização da importância da aplicação de *BI* no processo de tomada de decisão.

c) No melhor aproveitamento dos dados gerados pelas empresas e organizações públicas e privadas.

Quanto aos aspectos metodológicos, a natureza dessa pesquisa é um resumo de assunto. Quanto aos seus objetivos é uma pesquisa descritiva. Em relação aos procedimentos técnicos, é uma pesquisa bibliográfica e experimental.

2. Referencial Teórico

2.1. *Business Intelligence (BI)*

O termo *Business Intelligence (BI)* foi criado por Howard Dresner para definir métodos usados para melhorar o processo de tomada de decisão das empresas. O *BI* consiste em agrupar informações de diversas fontes e apresentá-las de forma unificada. Por se tratar de uma metodologia pode-se implementar *BI* com qualquer ferramenta de controle de dados, utilizando alguma linguagem de programação ou mesmo *frameworks* que auxiliam no processo. (BRAGHITTONI, 2017)

Um sistema de *BI* oferece aos tomadores de decisões informações e conhecimentos extraídos de dados, por meio da aplicação de modelos matemáticos e algoritmos. Em termos gerais, a adoção de um sistema de *BI* tende a promover uma abordagem científica e racional para a gestão de empresas e organizações complexas. (VERCELLIS, 2011)

O *BI* utiliza de modelos matemáticos, metodologias de análise e algoritmos para realizar o processo de obtenção de conhecimento. Algumas metodologias que podem estar presentes são: análise exploratória de dados, análise de séries temporais, implementação de aprendizado de máquina para a mineração de dados, entre outros. O BI serve para analisar os fenômenos acerca do negócio, isso significa que Business Intelligence precisa ser uma plataforma capaz não só de aglutinar as informações transacionais, mas também exibi-las de forma contextual. (BRAGHITTONI, 2017)

O primeiro grande conceito relacionado ao BI é de que as informações são cópias dos dados dos sistemas da sua empresa e, se necessário, de fontes externas. Mas o Business Intelligence, por definição, não deve gerar informações que não as de estatísticas sobre os dados importados das fontes chamadas transacionais. (BRAGUITTONI, 2017)

2.2. Sistema transacional e Sistema analítico

Sistema transacional é o sistema da empresa, onde os dados são gerados e atualizados. Sistemas em que as transações são geradas e atualizadas, como o Sistema de Pedidos, de chamados, CRM, entre outros. (BRAGHITTONI, 2017)

O Sistema analítico é o *BI* em si, para onde os dados são importados e onde as informações são disponibilizadas para que sejam analisadas. Determinante para tomar decisões efetivas e rápidas. (VERCELLIS, 2011)

2.3. Extract, Transform and Load (ETL)

O ETL descreve o processo completo em que uma empresa coleta todos os dados, estruturados e não estruturados, e gerenciados por qualquer número de equipes em todo o mundo, e os processa de forma que eles se tornem realmente úteis para fins comerciais. O ETL é uma forma importante de reunir todos os dados relevantes em um só lugar para torná-los práticos de analisar e permitir que executivos, gerentes, e outras partes interessadas tomem decisões de negócios mais fundamentadas com base neles. (RAUNTENBERG et al., 2019)

O processo de extração e transformação dos dados vindos do sistema transacional e de fontes externas, é a fase do processo em que regras são aplicadas para extrair e tratar os dados vindos do sistema transacional, o tratamento é feito para corrigir os erros mais frequentes, como por exemplo: duplicação de dados, a falta de dados (dados em branco), a existência de dados inadmissíveis e também a inconsistência entre dados armazenados em atributos diferentes e que possuem o mesmo significado. (BRAGHITTONI, 2017)

2.4. Data Warehouse (DW)

Como o próprio nome sugere, um data warehouse é o principal repositório para os dados disponíveis para o desenvolvimento de arquiteturas de *BI* e suporte. Podem armazenar dados internos, externos e pessoais, mais especificamente: os dados internos são aqueles de origem do sistema transacional da empresa, geralmente estão modulados, os externos têm origem de várias fontes, podendo ser emails, documentos, várias informações, e os pessoais são aqueles que muitas vezes são armazenados pelos responsáveis pela tomada de decisão. (Dresner, 1989).

O *DataWarehouse* é um banco de dados dimensional que possui um esquema em formato de estrela, esse formato é dado devido ao relacionamento entre as tabelas, permitindo assim uma melhora no tempo de respostas das consultas. Ele possui dois tipos de tabelas de dados: as tabelas Dimensões e as tabelas Fatos. As tabelas Dimensões representam as entidades que fazem parte dos processos de uma organização, são como as entidades de um banco de dados relacional comum. As tabelas Fatos representam transações ou processos que acontecem em uma organização. (RAUNTENBERG et al., 2019)

2.5. Online Analytical Processing (OLAP)

O *OLAP* corresponde a todo o conjunto de ferramentas destinadas a realizar as análises de *BI* e a exibir os dados para apoiar os processos de tomada de decisão. Pode usar de diversos modelos de armazenamento para apresentar as informações, que os seguintes modelos:

- ***MOLAP:*** Neste modelo, os dados são armazenados em cubos multidimensionais. Os cubos são bancos de dados multidimensionais que suportam análises dinâmicas. Eles são formados por mapas de acesso, informações detalhadas e informações agregadas. *MOLAP* é conhecido por oferecer ótimo desempenho em termos de velocidade de consulta, mas geralmente requer mais espaço de armazenamento devido à natureza multidimensional dos dados. (BRAGHITTONI, 2017)

- ***ROLAP****:* No modelo *ROLAP*, todos os dados são armazenados em um banco de dados relacional. Isso significa que os dados são armazenados em tabelas e podem ser acessados usando consultas *SQL*. O *ROLAP* tende a ter um tempo de resposta menor em comparação com o *MOLAP* para algumas consultas, mas pode ser menos eficiente para operações analíticas complexas.(BRAGHITTONI, 2017)
- ***DOLAP:*** *DOLAP* refere-se a sistemas *OLAP* que são executados em estações de trabalho ou desktops individuais. Geralmente, essas soluções são mais simples e voltadas para a análise local de pequenos conjuntos de dados. (BRAGHITTONI, 2017)

- ***HOLAP:*** O modelo *HOLAP* combina elementos do *ROLAP* e *MOLAP*. Os dados são armazenados tanto em um banco de dados relacional como em cubos multidimensionais. Isso permite a flexibilidade de usar a abordagem relacional quando necessário e, ao mesmo tempo, aproveitar as vantagens do modelo multidimensional para análises complexas. (BRAGHITTONI, 2017)

Cada modelo tem suas próprias vantagens e desvantagens, e a escolha do modelo adequado depende das necessidades específicas do projeto de BI, dos requisitos de desempenho e dos recursos disponíveis. É importante selecionar o modelo que melhor atende aos objetivos da análise e às limitações de infraestrutura.

3. Procedimentos metodológicos

Esta pesquisa segundo a sua natureza é um resumo de assunto, buscando sistematizar a área de conhecimento do projeto, indicando sua evolução histórica, e através das informações obtidas, levando ao entendimento de suas causas e explicações (WAZLAWICK, 2014).

Segundo seus objetivos é uma pesquisa descritiva. A pesquisa descritiva é mais sistemática, busca dados mais consistentes sobre determinado assunto, porém, não ocorre a interferência do pesquisador, que apenas expõe os fatos como realmente são (WAZLAWICK, 2014).

Em relação aos procedimentos técnicos, esta pesquisa é bibliográfica e experimental. A pesquisa bibliográfica implica o estudo de artigos, teses, livros e outras publicações usualmente disponibilizadas por editoras e indexadas, mas ela em si não produz qualquer conhecimento novo. Apenas supre o pesquisador de informações públicas que ele ainda não possuía. A pesquisa experimental caracteriza-se pela manipulação de um aspecto da realidade pelo pesquisador (WAZLAWICK, 2014).

De acordo com Wazlawick (2014), a pesquisa bibliográfica deve ser sistematizada seguindo estes passos:

a) Listar os títulos de periódicos e eventos relevantes para Business Intelligence e os títulos de periódicos gerais em computação que eventualmente possam ter algum artigo na área de Business Intelligence.

b) Obter a lista e todos os artigos publicados nos últimos cinco anos (ou mais) nesses veículos.

c) Selecionar desta lista aqueles títulos que tenham relação com Business Intelligence.

d) Ler o abstract desses artigos e, em função da leitura, classificá-los como relevância “alta”, “média” ou “baixa”.

e) Ler artigos de alta relevância e fazer fichas de leitura, anotando os principais conceitos e ideias aprendidos. Anotar também títulos de outros artigos possivelmente mencionados na bibliografia de cada artigo (mesmo que com mais de cinco anos) e que pareçam relevantes para o trabalho de pesquisa. Incluir esses artigos na lista dos que devem ser lidos (inicialmente o abstract e, se for relevante, o artigo todo).

f) Dependendo do caso, ler também os artigos de relevância média e baixa, mas iniciando sempre pelos de alta relevância.

Enquanto a pesquisa bibliográfica é elaborada a partir de materiais já publicados, a pesquisa experimental implica ter uma ou mais variáveis experimentais que podem ser controladas pelo pesquisador, e uma ou mais variáveis observadas, cuja medição poderá levar, possivelmente, à conclusão de que existe algum tipo de dependência com a variável experimental (WAZLAWICK, 2014).

Além disso, a pesquisa experimental deve utilizar rigorosas técnicas de amostragem e testes de hipóteses para que seus resultados sejam estatisticamente aceitáveis e generalizáveis (Barbetta, Reis & Bornia, 2004). A pesquisa experimental será estruturada da seguinte forma:

- Formulação do problema: **Como aplicar conceitos de Business Intelligence para otimizar a gestão de processos em telecomunicações empresarial?**
- Definição do plano experimental: foram extraídos dados do sistema transacional, os dados são referentes à telecomunicação da empresa que utiliza o sistema para gerenciar sua comunicação interna e, principalmente, a comunicação externa. Os dados extraídos foram tratados e carregados em um *Data Warehouse* para a criação dos dashboards informativos.
- Determinação do ambiente: o ambiente se trata de um servidor com sistema operacional Linux, no qual está implementado o sistema transacional, e um computador com sistema operacional *Windows* que contém todas as

configurações e ferramentas necessárias para a implementação do sistema de *BI*. São elas:

- *MySQL Workbench*: sistema de gerenciamento de banco de dados *MySQL* usado para conectar com o *MySQL* do sistema transacional.
- *SQL Server*: sistema de gerenciamento de banco de dados usado para criar o servidor que armazenará o *Data Warehouse*.
- *Python* e bibliotecas da linguagem: linguagem utilizada na criação do *script* que automatiza a implementação do processo de extração, transformação e carregamento (*ETL*) dos dados.
- *Power BI*: ferramenta de análise e visualização de dados usada para criar os *dashboards* informativos.

- Coleta de dados: Foi feita a instalação do *MySQL Workbench* para configurar a conexão com o sistema de banco de dados *MySQL* do sistema transacional. Após a conexão, a coleta de dados foi realizada através da execução de um *script Python* que conecta no banco de dados transacional e coleta os dados necessários.
- Análise e interpretação dos dados: Os dados coletados foram tratados e carregados em um *Data Warehouse* presente no computador do ambiente de testes. Por fim, o *Data Warehouse* é conectado em um projeto do *Power BI*, onde os dados foram utilizados na criação de tabelas, relatórios, gráficos e muitos mais em um *dashboard* interativo.
- Redação do relatório: Redação feita no TCC.

4. Motivação para a aplicação de Business Intelligence

Este capítulo tem como objetivo trazer à tona a motivação para aplicar *Business Intelligence* para obter conhecimento e auxiliar a gestão da telecomunicação de empresas. Para isso, foi feito um paralelo com um sistema utilizado por empresas para mostrar que, aplicando *BI* ao sistema em questão, tanto a obtenção de conhecimento quanto a gestão dos departamentos e da telecomunicação da empresa melhoraram.

O sistema transacional se trata de um *software* de comunicações unificadas de código aberto, que oferece configuração, gerenciamento e relatórios de telefonia baseados na *Web*. Explicando de uma forma mais precisa e detalhada, é um *software* que se comporta como uma central telefônica que recebe todas as ligações externas e internas e as envia para o destino pré-definido, unificando assim toda a telecomunicação da empresa. Além do funcionamento normal de uma central telefônica, o *software* disponibiliza: ferramentas para configuração do ambiente de telecomunicação empresarial, a possibilidade de incluir novas funções, e ferramentas para o gerenciamento da comunicação da empresa, como relatórios de ligações, gráficos informativos, entre outros.

No subcapítulo 4.1 está o detalhamento de uma empresa que utiliza um software de gerenciamento de telefonia e comunicação empresarial.

4.1 Hospital ABC

O Hospital ABC utiliza o sistema para implementar e gerenciar a comunicação tanto interna quanto externa entre seus diferentes departamentos. A solução para obtenção de relatórios oferecida pelo sistema disponibiliza relatórios com informações de cada ligação feita ou recebida, como por exemplo: o número de quem ligou, o id de quem atendeu a ligação recebida, se a ligação foi atendida ou não, entre outros. O principal problema em relação ao sistema é que a solução existente não é totalmente satisfatória Muitas vezes, essa solução não proporciona dados totalmente prontos para análise, o que requer que os responsáveis pelas tomadas de decisões tenham que realizar um trabalho adicional para extrair as informações necessárias, e que, muitas vezes não consigam extrair o que precisam. Além disso, a solução não fornece as métricas específicas e insights necessários

para atender às demandas individuais de cada empresa. Isso dificulta a avaliação efetiva do desempenho da comunicação e do atendimento da empresa, bem como a identificação de áreas que possam requerer melhorias.

Portanto, a aplicação de *Business Intelligence (BI)* no Hospital ABC tem como objetivo principal criar um ambiente no qual todos os dados, informações e métricas essenciais estejam prontamente disponíveis. Através da implementação do *BI*, a gerência do hospital terá acesso a uma visão completa e precisa de todos os aspectos relevantes para o funcionamento eficaz da instituição. Isso inclui dados detalhados, métricas personalizadas e *insights* especializados que são vitais para a tomada de decisões informadas e para garantir que a comunicação e o atendimento atendam aos padrões de qualidade desejados. Em última análise, a adoção do *BI* é uma iniciativa que visa aprimorar a eficiência e a eficácia dos processos do Hospital ABC, promovendo uma gestão mais embasada em dados e, como resultado, uma melhor qualidade de atendimento e satisfação do cliente.

5. Ferramentas utilizadas na implementação

Este capítulo tem como objetivo introduzir e explicar as funções das ferramentas empregadas durante a fase de implementação. Foram selecionadas e utilizadas ferramentas para a organização do ambiente de testes e a realização de testes do sistema de *BI*. Nesta seção, serão descritas detalhadamente as ferramentas, destacando suas finalidades e contribuições para o desenvolvimento do projeto:

5.1 *MySQL Workbench*

O *MySQL Workbench* é uma poderosa ferramenta visual de design de banco de dados que oferece um ambiente de desenvolvimento integrado para o sistema de banco de dados *MySQL*. Com o *MySQL Workbench*, os desenvolvedores podem realizar diversas tarefas, como por exemplo: modelagem de dados, desenvolvimento de consultas *SQL*, importação e exportação de dados, recursos de administração de bancos de dados, ferramentas de migração e modelos de relatórios.

No projeto do sistema de *BI* o *MySQL* foi empregado como ferramenta de visualização e de conexão com o banco de dados do sistema transacional, para que fosse possível a extração de dados do mesmo.

5.2 *SQL Server*

O *SQL Server* é um sistema de gerenciamento de banco de dados relacional desenvolvido e mantido pela *Microsoft*, sendo atualmente um dos principais bancos de dados do mercado. Com várias ferramentas integradas, é um sistema capaz de atender às demandas de uma ampla variedade de aplicações, desde projetos simples até ambientes que lidam com grande volume de dados.

O *SQL Server* conta também com uma ferramenta de interface gráfica chamada *SQL Server Management Studio (SSMS)*, que substitui o uso da ferramenta de linha de comando, tornando o trabalho com o banco de dados mais simples. Além disso, o *SQL Server* se destaca por sua robustez, escalabilidade,

segurança e suporte a recursos avançados, como armazenamento de dados em nuvem e integração com outras tecnologias da *Microsoft*.

No contexto do projeto do sistema de *BI*, o *SQL Server* foi escolhido como o sistema gerenciador de banco de dados responsável por abrigar o *Data Warehouse* que irá servir o produto final do projeto. Além de suas qualidades, um dos motivos da escolha do *SQL Server* é por ter uma boa conexão com a ferramenta de visualização e análise de dados, também da *Microsoft*, o *Power BI*.

5.3 *Python*

O *Python* é uma linguagem de programação amplamente adotada em diversas aplicações, incluindo desenvolvimento web, software, ciência de dados e *machine learning (ML)*. O *Python* é a escolha de muitos desenvolvedores devido à sua eficiência, facilidade de aprendizado e capacidade de ser executado em múltiplas plataformas. Uma das vantagens notáveis do *Python* é que o software relacionado pode ser baixado gratuitamente, o que o torna acessível a uma ampla gama de desenvolvedores. Além disso, o *Python* se integra bem com diferentes sistemas e acelera o processo de desenvolvimento, tornando-o uma linguagem de programação valiosa em uma variedade de cenários de aplicação.

No projeto do sistema de *BI*, o *Python* foi a linguagem escolhida para o desenvolvimento do *script* de automação do processo de *ETL*. Essa escolha se baseia nas capacidades versáteis do *Python*, que tornam a linguagem uma opção robusta e eficaz para a implementação do *ETL*, facilitando a manipulação e a transformação de dados, além de integrar-se bem com outras ferramentas e tecnologias frequentemente usadas em projetos de *BI*.

5.4 *VS Code*

O *Visual Studio Code* é um editor de código-fonte desenvolvido pela *Microsoft* para *Windows, Linux e macOS*. Ele inclui suporte para depuração, controle de versionamento *Git* incorporado, realce de sintaxe, complementação inteligente de código, *snippets* e refatoração de código. O *VS Code* foi usado como ambiente de desenvolvimento do código do *script*.

5.5 *Power BI*

O *Power BI* é uma plataforma unificada e escalonável para *Business Intelligence (BI)* empresarial e de autoatendimento. É a plataforma líder de mercado que combina uma experiência de usuário intuitiva com análises avançadas. É possível reunir dados para análise em segundos e utilizar esses dados para a criação de tabelas, gráficos, relatórios e muito mais. Os *dashboards* criados são interativos e podem ser compartilhados com as pessoas interessadas naqueles dados.

O *Power BI* foi escolhido para a visualização e análise dos dados finais, a sua escolha foi uma decisão motivada pela combinação de facilidade de uso, ampla capacidade de integração (diretamente com o *SQL Server*), recursos avançados para a criação de *dashboards* interativos e, principalmente, fáceis de interpretar, já que o objetivo final é disponibilizar os *dashboards* para os interessados na análise dos dados da empresa.

6. Implementação do sistema

Neste capítulo é detalhada a criação do ambiente de testes, realizando o projeto e o desenvolvimento do código e a implementação das ferramentas que irão compor o sistema de *BI* e serão utilizadas para as simulações. O objetivo da implementação é demonstrar a aplicação de *Python* para realizar os processos de tratamento dos dados e transferência dos dados entre os bancos do sistema transacional e analítico, e também mostrar o desenvolvimento dos *dashboards* na ferramenta *PowerBI*.

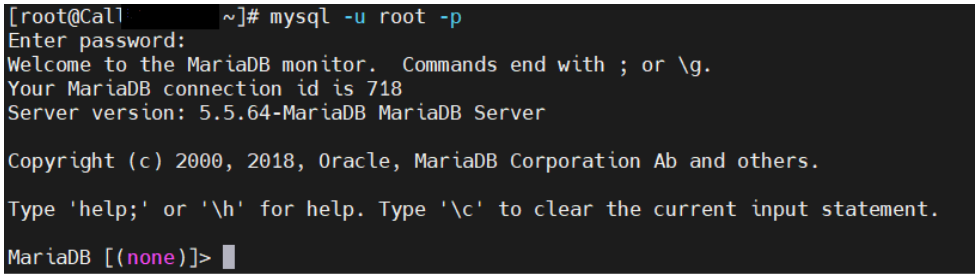
6.1 Conexão com o banco do sistema transacional

Para que seja possível realizar o processo de extração dos dados armazenados no software de telefonia e comunicação, ou seja, o sistema transacional, é preciso estabelecer uma conexão entre o ambiente em que as demonstrações e testes ocorrerão e o banco de dados do sistema. Para estabelecer a conexão foi utilizado o *MySQL Workbench*, ferramenta que, como visto na seção 5.1, permite a realização de inúmeras atividades, entre elas a conexão com bancos presentes em servidores remotos.

Para estabelecer uma conexão com o banco de dados do sistema transacional, é necessário seguir um procedimento inicial que envolve a concessão de permissões no *MySQL*. A concessão de permissões é necessária para que um usuário específico acesse o banco de dados remotamente. Essa ação é realizada acessando diretamente o servidor em que o sistema transacional está em execução, utilizando as credenciais de administrador *root*, o superusuário com privilégios administrativos que possui controle sobre o sistema de gerenciamento do *MySQL*. Ao acessar o *MySQL* como *root*, é possível configurar quais usuários ou *hosts* remotos terão permissão para se conectar ao banco de dados, podendo especificar os privilégios de acesso, incluindo permissões para leitura, gravação, criação de tabelas, entre outras operações. O acesso é feito em duas etapas: primeiro o comando “mysql -u root –p" é usado para iniciar a sessão do cliente *MySQL* no terminal, e, explicando de forma mais detalhada: o comando “mysql” é o comando principal que inicia a linha de comando do cliente *MySQL*, permitindo que você interaja com o servidor, o argumento “-u” é usado para especificar o nome de

usuário que você deseja usar para a conexão, que neste caso é “root”, e por último o argumento "-p" instrui o cliente *MySQL* a solicitar a senha do usuário após a execução do comando, que seria a segunda etapa do acesso ao *MySQL*. O procedimento todo pode ser visto na figura 1:

Figura 1 - Acesso ao *MySQL* do sistema transacional

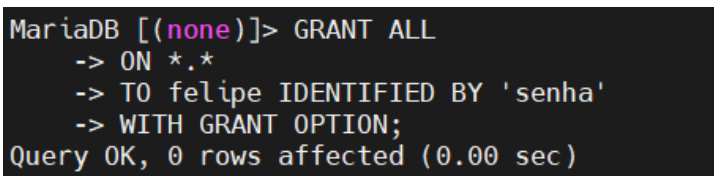


Fonte: Autoria própria

Após a conclusão do acesso, o comando “GRANT ALL” é executado no terminal. Ao executar esse comando, o cliente *MySQL* solicita os parâmetros necessários para realizar a concessão de privilégios, a ordem dos parâmetros é a seguinte:

1. **Nome da Base de Dados**: Neste caso, como a concessão foi feita para todas as bases de dados, portanto, o primeiro parâmetro é preenchido como " *. * " para indicar todas as bases de dados disponíveis no servidor.
2. **Nome de Usuário**: O segundo parâmetro é o nome do usuário que receberá as permissões.
3. **Senha do Usuário**: O terceiro parâmetro é a senha que será associada ao usuário para autenticação ao acessar o banco de dados.

Figura 2 - Concedido permissões ao novo usuário

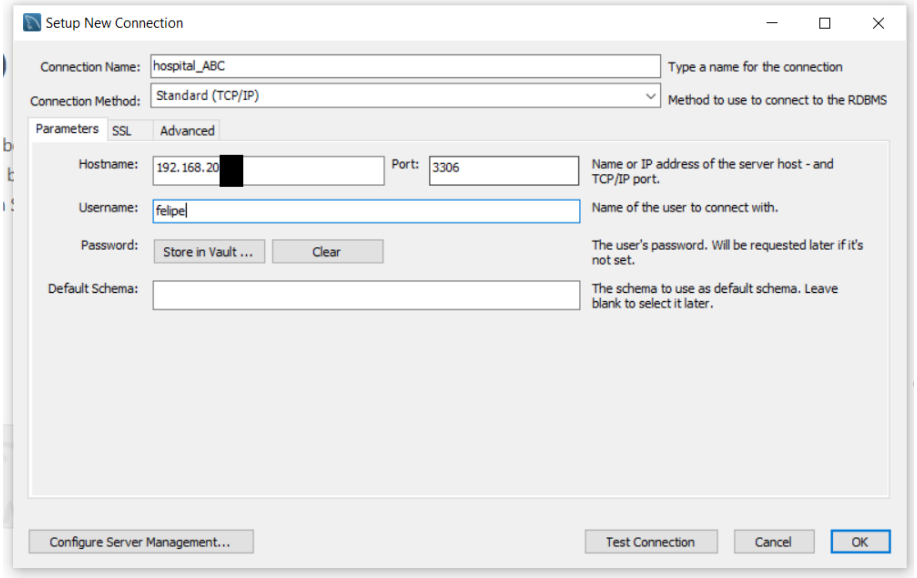


Fonte: Autoria própria

A figura 2 mostra a sintaxe do comando (seus valores de usuário e senha correspondem a valores representativos).

Após conceder as permissões de usuário, o próximo passo foi criar a conexão no *Workbench.* Para estabelecer a conexão, algumas etapas foram seguidas. Primeiramente, foi atribuído um nome à conexão, em seguida, o método de conexão foi definido, e por fim, os dados do servidor ao qual a conexão seria estabelecida foram fornecidos em duas etapas. Os dados do servidor são: o hostname do servidor, a porta utilizada para a conexão, o nome de usuário com as permissões concedidas e, posteriormente, a senha previamente definida. A figura 3 mostra a interface do *Workbench* para o inserção dos dados de conexão com o banco de dados remoto, é possível perceber que a senha para a conexão do usuário com o banco ainda não será inserida.

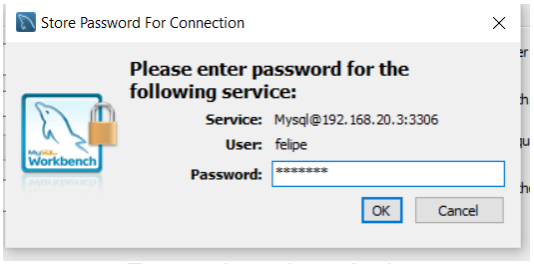
Figura 3 - Criação da conexão no MySQL Workbench



Fonte: Autoria própria

A senha é preenchida em outra interface que é apresentada após o preenchimento dos botões, mostrada na figura 4 e que é acionada após o usuário selecionar o botão *Store in Vault* presente na interface anterior.

Figura 4 - Armazenando a senha para conexão



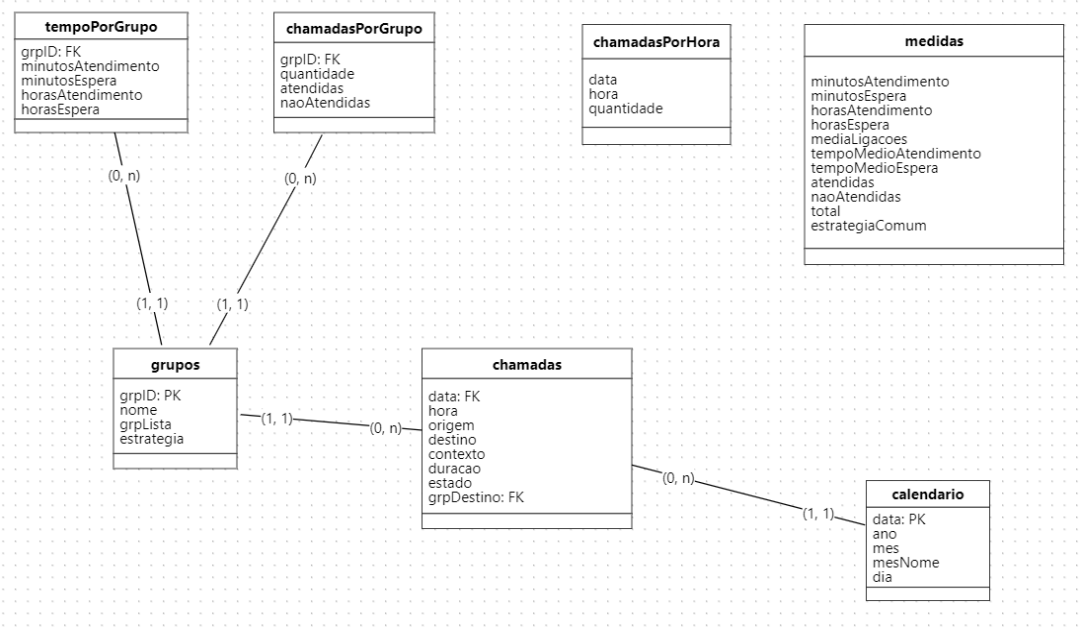
Fonte: Autoria própria

Após o preenchimento dos dados, basta testar a conexão com o *MySQL* do servidor remoto e acessar sua interface visual do sistema, onde é possível acessar seus *schemas*, bancos e tabelas.

6.2 Construção do *Data Warehouse*

O *Data Warehouse (DW)* será o repositório dos dados que forem extraídos do sistema transacional. Para otimizar a estrutura de armazenamento de dados, o *DW* foi projetado tendo em conta as características técnicas essenciais, como a eficiência de consulta e a integridade dos dados, entre outras considerações. Além disso, foi fundamental projetar as tabelas e colunas do *DW* pensando em armazenar todos os dados necessários para uma análise completa, a fim de suportar a geração de relatórios, gráficos e tabelas no momento da sua criação.

Figura 5 - Modelo lógico do DW



Fonte: Autoria própria

O armazenamento e a organização desses dados também foram planejados com o intuito de facilitar a sua utilização posterior no *Power BI*. Considerando as necessidades levantadas, o *DW* foi projetado de acordo com o modelo lógico mostrado na figura 5. A figura 5 detalha o modelo lógico do *DW*, com as entidades, os atributos e os relacionamentos entre entidades já estruturados da forma como realmente serão implementados. A seguir está o modelo lógico do *DW*, contendo algumas informações a mais sobre a organização lógica do *DW*:

1. **Tabela Fato:**
    - **chamadas:** armazena os dados mais importantes correspondentes às ligações registradas pelo sistema. Por registrar o evento principal que será analisado pelo sistema de *BI*, é a tabela Fato.
2. **Tabelas Dimensões:**
    - **grupos:** armazena informações relacionados aos grupos de chamada existentes no sistema, cada grupo de chamada é um departamento independente que recebe e realiza chamadas.
    - **chamadasPorGrupo:** armazena as chamadas totais, as chamadas atendidas e as chamadas não atendidas por cada grupo (departamento).
    - **chamadasPorHora:** armazena a quantidade de chamadas registradas no sistema a cada hora do dia para cada dia filtrado e presente na tabela calendario.
    - **tempoPorGrupo:** armazena a quantidade de horas e minutos que cada grupo passou em atendimento e com suas chamadas em espera.
3. **Tabelas auxiliares:**
    - **calendario:** é uma tabela criada através dos dados extraídos do sistema transacional, o intervalo de datas presentes na tabela calendario é o mesmo intervalo de datas das ligações filtradas do sistema transacional.
    - **medidas:** a tabela medidas armazena informações que serão utilizadas na criação dos *dashboards*, principalmente como métricas para avaliar o desempenho e também como uma

forma de facilitar a combinação de valores para formar novos gráficos.

As tabelas auxiliares são tabelas criadas para auxiliar na criação dos relatórios para análise. Tabelas calendário e medidas são muito utilizadas nos sistemas de *BI*, e a estrutura da tabela medidas acaba variando de um sistema de *BI* a outro, mas um sistema de *BI* pode contar com várias tabelas auxiliares além destas.

Como descrito anteriormente, o *SQL Server* foi a ferramenta escolhida para abrigar o *DW* e o *SQL Server Management Studio (SSMS)* a ferramenta de interface gráfica utilizada para realizar as operações desde a criação do *DW* até a criação das tabelas. Para acessar o servidor criado durante a instalação do *SQL Server* basta abrir o *SSMS* e selecionar qual servidor será usado e as credenciais que serão usadas para acessar o servidor (pode ser as próprias credenciais de usuário do *Windows*). Assim que estiver conectado já é possível criar um banco para ser o *DW* e assim criar as tabelas. Para criar as tabelas que irão compor o *DW* e que serão carregadas com dados pelo *script Python*, foi utilizado o código *SQL* mostrado nas figuras 6 e 7:

Figura 6 - Código *SQL* para criação do *DW* 1

```
CREATE TABLE grupos (
    grpNum VARCHAR(10),
    estrategia VARCHAR(50),
    grpLista VARCHAR(255),
    nome VARCHAR(255),
    PRIMARY KEY(grpNum)
);

CREATE TABLE chamadas (
    data DATE,
    hora TIME(0),
    origem VARCHAR(20),
    destino VARCHAR(20),
    contexto VARCHAR(50),
    duracao VARCHAR(10),
    estado VARCHAR(255),
    grpDestino VARCHAR(10)
    FOREIGN KEY (grpDestino) REFERENCES grupos(grpNum) ON DELETE CASCADE,
    FOREIGN KEY (data) REFERENCES calendario(data) ON DELETE CASCADE
);

CREATE TABLE chamadasPorGrupo (
    grpID VARCHAR(10),
    quantidade int,
    atendidas int,
    naoAtendidas int,
    FOREIGN KEY (grpID) REFERENCES dGrupos(grpNum) ON DELETE CASCADE
);
```

Fonte: Autoria própria

Figura 7 - Código *SQL* para a criação do *DW* 2

```sql
CREATE TABLE medidas (
    minutosAtendimento int,
    minutosEspera int,
    horasAtendimento float,
    horasEspera float,
    mediaLigacoes int,
    tempoMedioAtendimento float,
    tempoMedioEspera float,
    atendidas int,
    naoAtendidas int,
    total int,
    estrategiaComum VARCHAR(50)
);
CREATE TABLE tempoPorGrupo (
    grpID VARCHAR(10),
    minutosAtendimento int,
    minutosEspera int,
    horasAtendimento float,
    horasEspera float,
    FOREIGN KEY (grpID) REFERENCES grupos(grpNum) ON DELETE CASCADE
);
CREATE TABLE chamadasPorHora (
    data DATE,
    hora TIME(0),
    quantidade int
);
CREATE TABLE calendario (
    data DATE,
    ano int,
    mes int,
    mesNome VARCHAR(25),
    dia int,
    PRIMARY KEY(data)
);
```

Fonte: Autoria própria

6.3 Projeto e desenvolvimento do *script Python*

O objetivo de utilizar um script feito em *Python* para realizar o processo de extração, transformação e carregamento dos dados (*ETL*) foi automatizar essa etapa da construção de um sistema de *BI*. Uma vez que o *ETL* envolve a manipulação e tratamento dos dados, é crucial otimizar e inspecionar essa etapa para evitar erros no processo de tratamento dos dados. Isso também se aplica à extração de todos os dados relevantes e ao carregamento adequado dos mesmos, garantindo assim a obtenção dos dados necessários para o projeto. O *script* foi projetado para cumprir com as funcionalidades de:

- Realizar a conexão com o banco de dados do sistema transacional e com o *Data Warehouse* do sistema de *BI*.
- Extrair os dados selecionados do sistema transacional.
- Realizar operações de transformação e manipulação de dados para gerar novos dados e informações através dos dados extraídos.
- Alimentar o *Data Warehouse* com os dados extraídos e os novos dados gerados.

O *Python* foi escolhido como a linguagem para desenvolvimento do *script* devido ao grande número de bibliotecas e a facilidade de implementar tratamento de dados utilizando a linguagem.

O *script* em *Python* desenvolvido apresenta uma arquitetura bem organizada, sendo estruturado em módulos que desempenham funções específicas e são interligados para atingir os objetivos do projeto. Os principais módulos que compõem o *script*: ***connect***, ***insertions***, ***selections***, ***data_transform*** e ***main:***

- **Módulo *connect*:** responsável por estabelecer conexões com o banco de dados transacional e o *Data Warehouse*, garantindo a interação com as fontes de dados e acesso às informações essenciais para o processamento.
- **Módulo *insertions*:** contém as funções de inserção dos dados que foram extraídos diretamente do banco de dados transacional (os novos dados que foram gerados pelo script são inseridos em outro módulo).
- **Módulo *selections*:** contém as operações de consulta de dados, ou seja, as funções que serão utilizadas por todo o *script* para extrair dados específicos tanto do sistema transacional quanto do sistema de *BI* para a utilização dos mesmos por outros métodos. É um módulo crucial para a obtenção de dados através da reutilização de código.
- **Módulo *data_transform*:** se trata do módulo essencial para o tratamento dos dados extraídos e para a transformação desses dados em novos dados e novas informações que serão

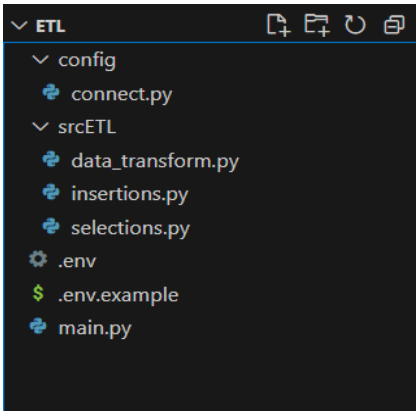
importantes na análise futura. Ele abriga funções responsáveis por processar os dados de acordo com as exigências do projeto.

- **Módulo *main*:** é o ponto de entrada do *script*, é onde o fluxo de execução do programa se inicia para que as funções dos outros módulos sejam chamadas.

O *script* foi projetado e desenvolvido em conformidade com as melhores práticas de programação orientada a funções, garantindo uma estrutura modular que facilita a manutenção e a extensão do código. Cada módulo desempenha um papel específico e contribui para a eficiência e clareza do *script* como um todo. Isso torna o código mais organizado, reutilizável e fácil de compreender, permitindo a realização eficiente do processamento e análise de dados, além de assegurar uma base sólida para futuras expansões e melhorias.

Seguindo para o desenvolvimento do *script*, utilizando o *VS Code*, a etapa inicial do foi criar a estrutura de diretórios que irão organizar o *script*, é importante começar pela organização estrutural para que o processo de desenvolvimento seja organizado desde o início. Pensando em agrupar os módulos que possuem funções semelhantes em um mesmo diretório, a estrutura final ficou da forma que está relatada na figura 8:

Figura 8 - Estrutura do *script Python*



Fonte: Autoria própria

A figura 8 mostra a organização do projeto, é possível ver que o diretório raíz do projeto é dividido em dois sub diretórios: *config* e *srcETL*. O diretório *config* armazena o módulo *connect.py* e os arquivos *.env* e *.env.example*, o *.env* é o

arquivo responsável pela configuração do ambiente e do projeto em si, o módulo connect.py é responsável pela conexão com os bancos de dados, o arquivo .env que contém as variáveis de ambiente que são usadas na conexão com os bancos (o arquivo .env.example não contém os valores das variáveis de ambiente, e somente é usado como uma forma de exemplo do arquivo .env). O diretório srcETL armazena os módulos que incluem as funções de extração, transformação e carga dos dados, sendo eles os módulos: *data_transform.py*, *insertions.py* e *selections.py*. Além dos módulos que estão nos subdiretórios, no diretório raíz ainda está o módulo principal *main.py*, responsável por chamar as funções dos outros módulos. O código correspondente ao módulo *connect.py* pode ser visualizado no apêndice A.

No módulo *connect.py* foram utilizadas as bibliotecas *pyodbc*, *os* e *dotenv*. É possível ver que os parâmetros de conexão não estão sendo diretamente inseridos, por se tratarem de informações sensíveis os valores são substituídos por variáveis de ambiente, os verdadeiros valores estão em *.env*. Para utilizar os parâmetros de conexão é preciso criar variáveis que recebam esse valores através da biblioteca os e do método *load_dotenv* herdado da biblioteca *dotenv*. Após receber os valores, a *string* de conexão é criada utilizando a biblioteca *pyodbc* e a função *connect* que irá estabelecer a conexão do *script* com os bancos é chamada, criando assim os objetos de conexão e por consequência os objetos chamados cursores, que serão chamados nos outros módulos para realizarem as operações de inserção e seleção nos bancos. O arquivo *.env* é semelhante ao *.env.example* mostrado no apêndice B.

O arquivo *.env* é uma ótima prática para a manipulação de variáveis de ambiente, principalmente por disponibilizar as variáveis para serem usadas por todo o programa e também por manter valores sensíveis sem serem expostos.

Após o desenvolvimento do módulo de conexão, o segundo módulo a ser desenvolvido foi o módulo *selections.py*. O módulo *selections* corresponde a duas funções, uma função extrai dados do banco do sistema transacional e a outra extrai dados do próprio *Data Warehouse*. A extração de dados do *Data Warehouse* quando for necessário transformar um dado que já tenha sido transportado para lá é bastante útil pois muitas consultas no banco do sistema transacional podem sobrecarregar as operações do sistema, portanto sempre que possível, é melhor consultar o *Data Warehouse*. O módulo *selections.py* é retratado no apêndice C.

É possível visualizar a importação do módulo *connect.py* para que seja possível usar as funções de conexão criadas dentro do módulo *connect.py* e assim extrair os dados necessários.

O módulo seguinte é o módulo *insertion.py*, e como pode ser visto na figura 12 este módulo é composto por duas funções: *insertGroup* e *insertCalls*, a primeira realiza inserção na tabela Dimensão grupos e a segunda na tabela Fato chamadas. Como dito anteriormente, é o módulo correspondente às funções de inserção de alguns dados no *DW*. Os dados inseridos no *DW* a partir das funções *insertGroup* e *insertCalls* são os dados extraídos do banco transacional e diretamente inseridos no *DW*, sem que seja necessário para por outras operações. Portanto, aqueles dados que são extraídos, tratados e que muitas vezes foram usados para gerar novos dados pertinentes, são inseridos utilizando outras funções que estão em outro módulo que será tratado mais a frente. A ideia de dividir a inserção de dados dessa forma foi para que ficassem mais modularizados e divididos com relação a sua função, para facilitar a manutenção do código e alterações futuras. O código das duas funções que formam o módulo insert.py pode ser visualizado no apêndice D.

Seguindo o desenvolvimento, o próximo módulo a ser tratado é o módulo *data_transform.py*, nele contamos com mais algumas funções, que dessa vez, além da inserção de dados, realizam tratamento de dados e produção de novos dados através dos extraídos. As funções e suas respectivas funcionalidades são:

- ***date:*** transforma a coluna *calldate* do banco transacional, que possui data e hora de cada ligação, em um dado com somente a data da ligação, retornando uma variável com essa data.
- ***hours:*** retorna uma variável com a parte referente a hora da coluna *calldate.*
- ***minutesToHours:*** função auxiliar usada por outras funções para transformar minutos em horas, realiza cálculos que transformam um valor inteiro em minutos no seu valor correspondente em horas.
- ***callsByGroup:*** através dos dados extraídos calcula a quantidade total de ligações recebidas e a quantidade de ligações atendidas e não atendidas por cada grupo. Insere os dados da tabela *callsByGroup*.

- ***hoursByGroup:*** através dos dados extraídos calcula quanto tempo cada grupo passou em atendimento ou com as ligações em espera, tanto em minutos quanto em horas. Insere os dados da tabela *hoursByGroup.*
- ***callsByHour:*** calcula a quantidade de ligações recebidas por cada hora dos dias filtrados. Insere os dados da tabela *callsByHour.*
- ***createCalendar:*** através dos registros das datas das ligações extraídas é montada a tabela calendário.
- ***createMeasures:*** chama as funções average, *totalHours*, *totalCalls* e *maxStrategy* e insere os dados retornados na tabela de medidas.
- ***average*****:** calcula e retorna a média de ligações registradas por dia.
- ***totalHours*****:** calcula e retorna a quantidade de minutos e horas de atendimento e de espera levando em conta todas as ligações extraídas do sistema transacional.
- ***totalCalls*****:** calcula e retorna a quantidade de ligações atendidas, não atendidas e o total de ligações. Diferente da função *callsByGroup*, esse cálculo obtém o valor geral.
- ***maxStrategy*****:** através dos dados extraídos verifica a estratégia de ligação mais usada e a retorna.

O apêndice E mostra detalhadamente como são as funções e as técnicas usadas para desenvolvê-las e obter os resultados desejados para a utilização dos dados.

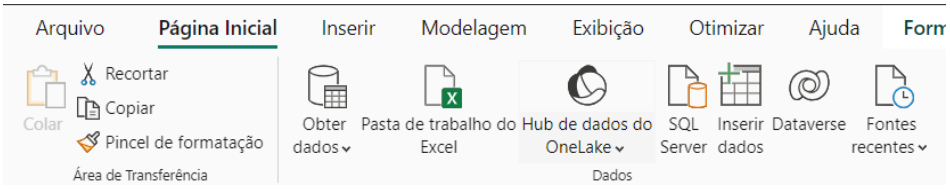
O último módulo desenvolvido foi o *main.py*, esse não se trata de um módulo como os outros que são formados por funções, é um módulo que marca o início do fluxo de execução do programa, ou seja, nele estão as chamadas para as funções dos outros módulos serem executadas. As chamadas seguem uma ordem de dependência, pois algumas funções dependem de outras que também são chamadas pelo módulo. O apêndice F mostra o módulo *main.py* e as chamadas que o compõem.

6.5 Criação dos *dashboards*

Após o desenvolvimento do *script* responsável pela etapa de *ETL* do *BI*, o primeiro passo para o desenvolvimento das etapas de análise e visualização dos dados para a obtenção de conhecimento é executar o *script* para que o processo de *ETL* seja feito e o *DW*, já estruturado anteriormente no *SQL Server*, seja carregado pelos dados extraídos e os novos dados gerados pelo *script*.

A próxima etapa executada foi a importação dos dados para o *Power BI*, onde eles serão analisados para a criação dos *dashboards* interativos e posteriormente serão visualizados pelo responsável por analisar e tomar decisões. Por se tratarem de ferramentas desenvolvidas pela *Microsoft*, a importação dos dados do *SQL Server* para o *Power BI* é bem simples, basta clicar na opção *SQL Server* na barra superior da página inicial da ferramenta, que é aberta quando se inicia um novo projeto. A figura 9 mostra com detalhes a seleção do compartilhamento:

Figura 9 - barra de opções do *Power BI*



Fonte: Autoria própria

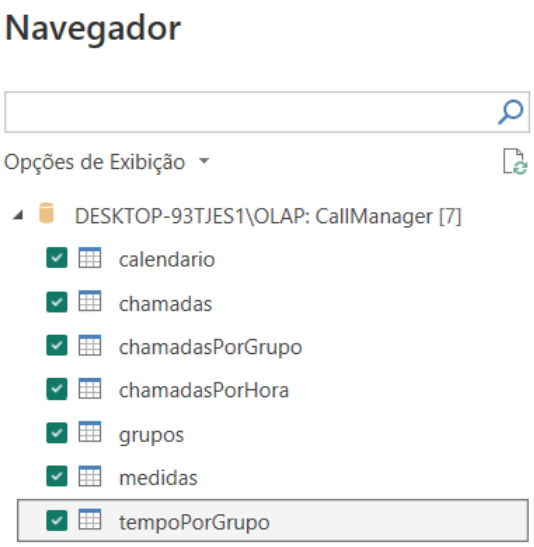
Após isso, o processo de importação acontece em duas etapas: a primeira é inserir o *host* do servidor onde o *DW* está e o nome do próprio *DW*, após isso basta selecionar as tabelas que serão importadas e finalizar a importação dos dados.

Figura 10 - Seleção do servidor *SQL Server*

Fonte: Autoria própria

A figura 10 mostra a etapa de inserção dos parâmetros de servidor para se conectar ao *SQL Server*. A figura 11 mostra a etapa de seleção das tabelas do *DW* para a importação ao *Power BI*:

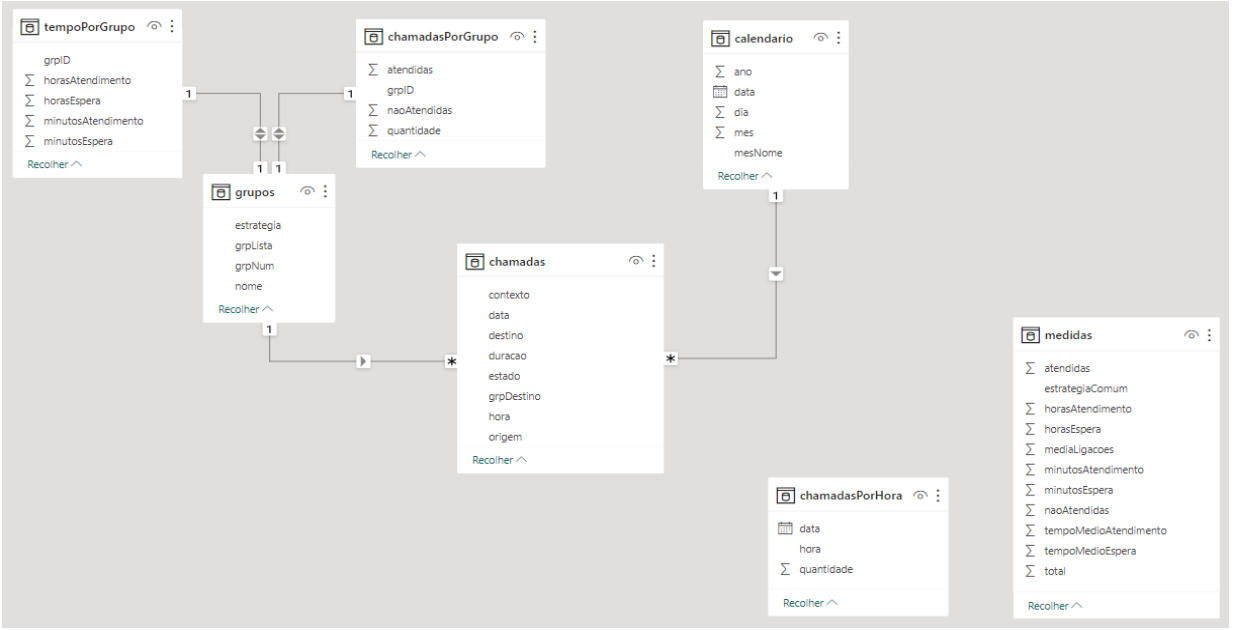
Figura 11 - Seleção das tabelas a serem importadas



Fonte: Autoria própria

Após importar os dados é possível visualizar um modelo relacional dos dados importados em forma de tabelas, podendo verificar também seus relacionamentos, como na figura 12:

Figura 12 - Modelo relacional das tabelas importadas



Fonte: Autoria própria

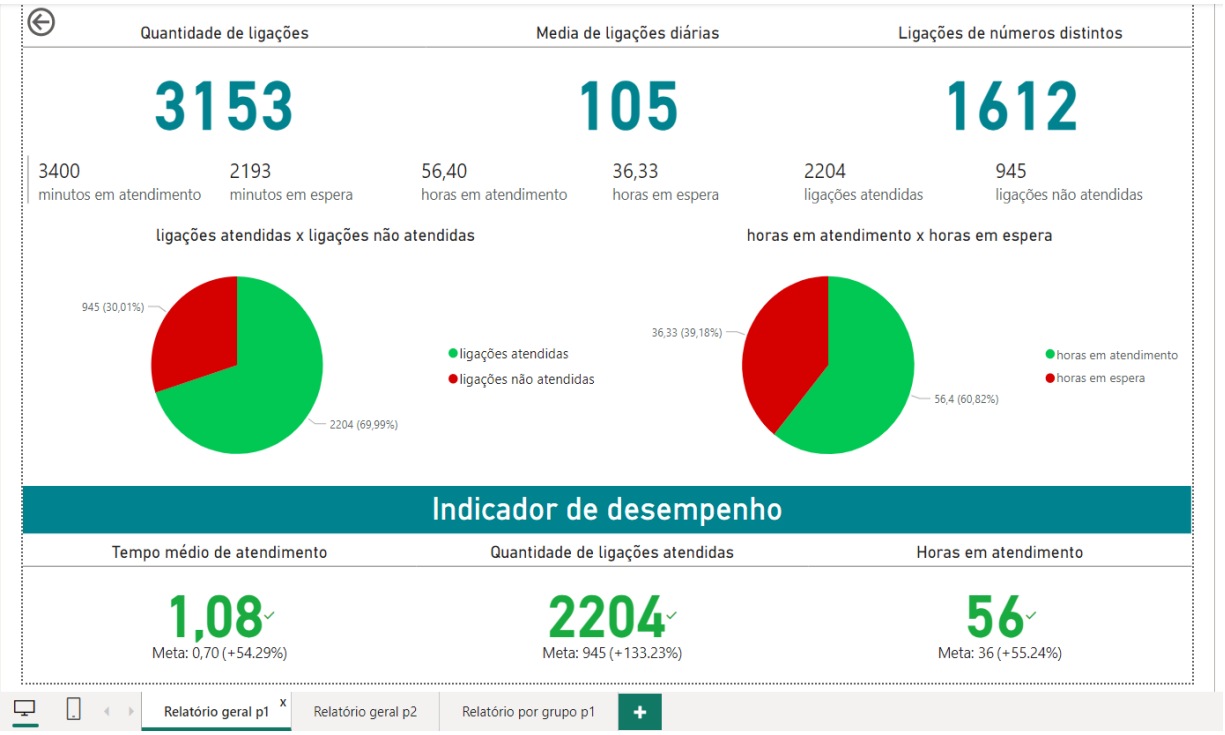
Com os dados importados ao Power BI, o próximo passo importante foi analisar os dados para definir quais ferramentas (relatórios, gráficos, cartões informativos, entre outros) irão compor os nossos dashboards, levando em consideração o objetivo final de obter conhecimento para poder qualificar o funcionamento da telecomunicação de uma empresa. A importância em definir o objetivo e de analisar os dados que serão expostos se dá pelo fato de que assim é possível escolher as melhores formas de mostrar esses dados, as melhores formas de relacionar os dados para gerar informação, e também a melhor forma de desenvolver um *dashboard* simples, fácil de ser entendido e que cumpra seu objetivo informativo e interativo. A criação dos gráficos de cada *dashboard* é feita manualmente, simplesmente arrastando para o gráfico os dados importados anteriormente que irão ser relacionados naquele gráfico.

Portanto, para seguir a idéia de manter a clareza e objetividade na etapa de análise e obtenção de informação, o *dashboard* foi dividido em três páginas: duas páginas se encarregam de disponibilizar as informações que são comum a todo o sistema, ou seja, as informações que mostram o funcionamento da telecomunicação da empresa como um todo, e a terceira página disponibiliza informações extraídas por cada grupo (departamento) da empresa.

A primeira página é composta por cartões informativos, gráficos de setores e medidas de desempenho. A ideia de incluir os cartões informativos e as medidas de desempenho na primeira página foi concentrar as informações principais e mais importantes logo no início e também de uma forma intuitiva direta. Os cartões contém medidas diretas como número total de ligações, média de ligações por dia (no período filtrado), horas totais em atendimento, entre outros valores que também foram obtidas da tabela medidas gerada no processo de *ETL*. Além disso, algumas medidas importantes para verificar o desempenho da empresa foram demonstradas através de *KPI's*, ou seja, métricas de desempenho, como por exemplo: em uma empresa podemos assumir que o número de ligações atendidas deve ser sempre maior que o número de ligações não atendidas, sendo assim o valor a ser medido é o número de ligações atendidas e a meta a ser batida é o valor de ligações não atendidas, utilizando a ferramenta de *KPI's* é possível verificar se o desempenho

está positivo ou negativo em relação a meta. Para finalizar, os gráficos de setores foram uma escolha segura para relacionar os valores “ligações atendidas x ligações não atendidas” e “horas em atendimento x horas em espera”, por se tratar de um gráfico de fácil entendimento e que pode diminuir o tempo de análise para a obtenção de informações. A figura 13 mostra o resultado final da primeira página do relatório:

Figura 13 - *Dashboard* geral 1



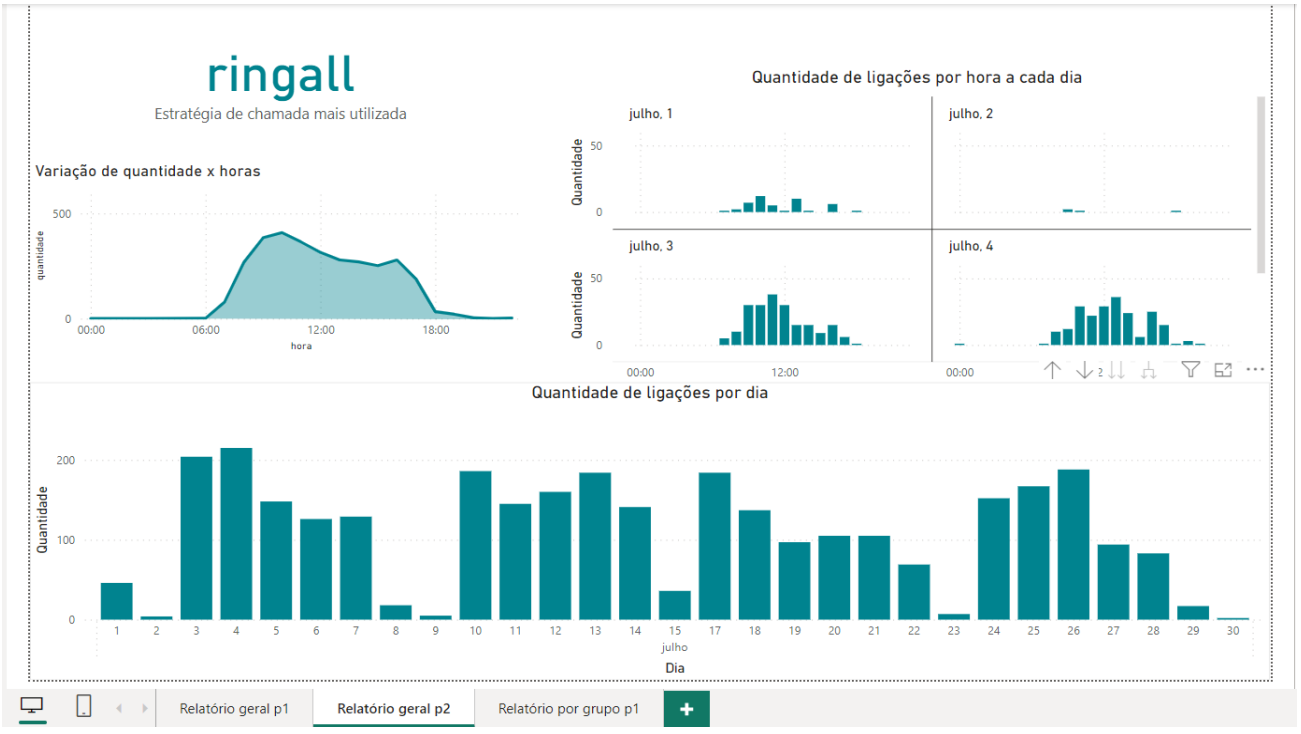
Fonte: Autoria própria

A Partir da análise do primeiro *dashboard* já é possível obter algumas informações para tomar decisões futuras, claro que irá depender do objetivo e métricas de cada empresa, mas com essa análise é possível identificar alguns pontos, como por exemplo:

- Se as ligações estão ficando muito tempo em espera, oque pode indicar que existem poucos atendentes para o fluxo de ligações.
- Se a empresa está tendo muitas ligações perdidas ou se está batendo a meta de atendimento, como acontece no exemplo da figura 13, em que a meta está sendo batida em +133 %.
- Um gerente pode decidir aumentar a média de ligações diárias através da divulgação dos canais de atendimento da empresa.

A segunda página apresenta os dados com marcos temporais, oque significa que a tabela calendario também foi usada com o para relacionar os dados com medidas de tempo, como por exemplo dias e horas. Além da tabela calendario, a tabela chamadasPorHora também é usada, pois ela já foi gerada durante a criação do *DW* para armazenar dados relacionados aos dias e as horas de cada dia. Para representar os dados, foram usados gráficos de colunas e um gráfico de área. A ideia dos gráficos de colunas é bem simples, um gráfico mostra a quantidade de ligações registradas por cada dia durante o mês filtrado, o outro gráfico mostra a quantidade de ligações registradas por cada hora do dia em todos os dias. O gráfico de área é semelhante, ele utiliza um somatório da quantidade de ligações registradas a cada hora em toda a tabela chamadasPorHora para chegar a uma média de todos os dias e identificar os horários em que o fluxo de ligações é maior.

Figura 24 - *Dashboard* geral 2



Fonte: Autoria própria

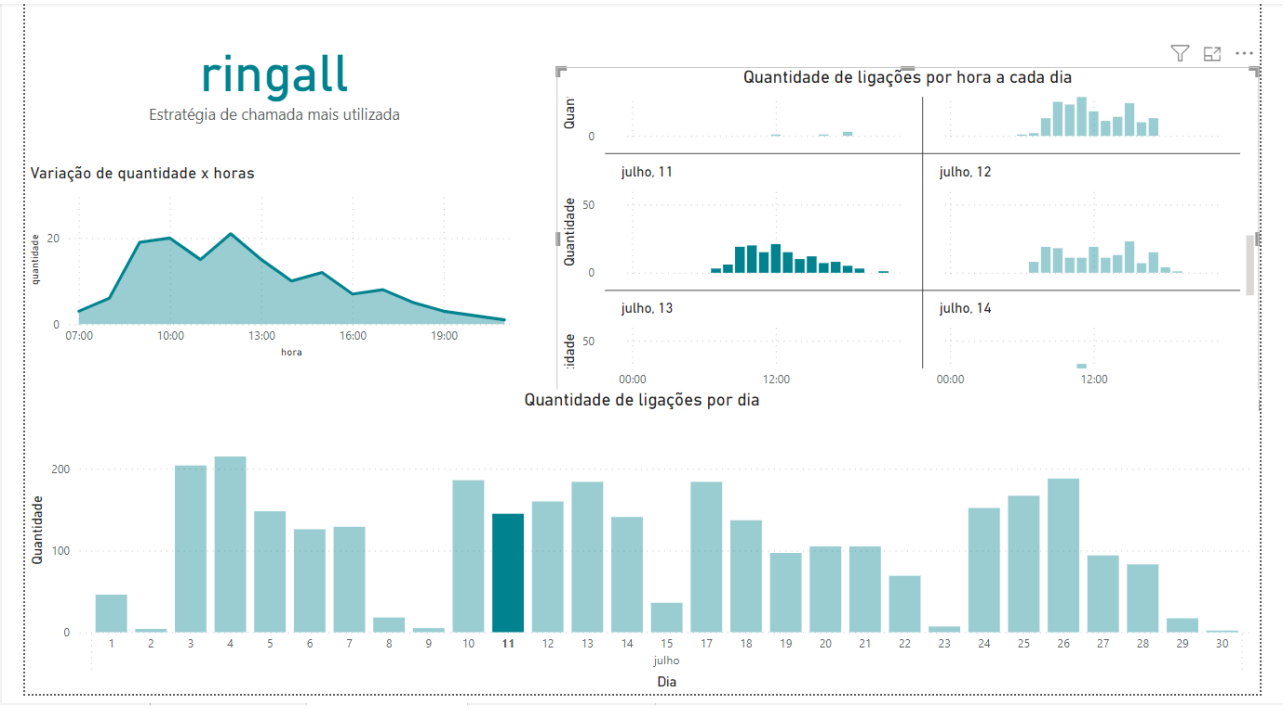
A segunda página conta também com um cartão de visualização sobre a estratégia de chamada mais utilizada pelo sistema, isto é, como cada grupo do sistema transacional é programado para receber chamadas, neste caso a estratégia mais usada é a *ringall*, o que significa que cada ligação recebida é encaminhada para todos os atendentes disponíveis para atendê la.

Através da análise da segunda página a obtenção de conhecimento acerca do funcionamento da telecomunicação da empresa se torna muito mais profunda, permitindo identificar padrões como:

- Horas do dia em que o fluxo de ligações é maior, permitindo manejar atendentes para aqueles horários.
- Dias da semana ou do mês que possuem um padrão de receber muitas ligações para alocar uma equipe maior para tais dias.
- Verificar como o fluxo de ligações se comporta mensalmente a cada horário, para se planejar a longo prazo e fazer alterações no atendimento baseado no fluxo identificado.
- Através dos marcos temporais é possível obter informações mais completas, pois o tempo é um identificador muito importante para constituir uma informação correta e é um diferencial na determinação do que é uma informação valiosa ou não.

A segunda página é o primeiro dos *dashboards* em que é possível interagir selecionando as colunas dos gráficos. Visualizando a figura 15, ao selecionar a coluna de um gráfico, os outros dois sofrem alterações de acordo com a coluna selecionada, por exemplo ao selecionar a coluna referente ao dia 11/07 do gráfico “Quantidade de ligações por dia”, a coluna selecionada ganha um destaque de cor em relação às outras, o gráfico “Quantidade de ligações por hora a cada dia” também receberá um foco somente no dia 11 de julho, e o gráfico de área terá sua área alterada para a área correspondente ao dia selecionado. Essa interação do usuário com os gráficos é muito interessante e torna a análise muito mais dinâmica e até mesmo mais didática, pois, através dessa função é possível analisar espaços de tempo menores, também é possível focar melhor em cada dia para identificar informações importantes, além de que torna bem menos complexo a criação de gráficos permitindo a reutilização de um mesmo gráfico para mostrar informações variadas, já que um mesmo gráfico pode alterar seu conteúdo para mostrar diferentes valores dependendo do dia selecionado, como acontece no gráfico de área do *dashboard* em questão. As alterações podem ser vistas na figura 15:

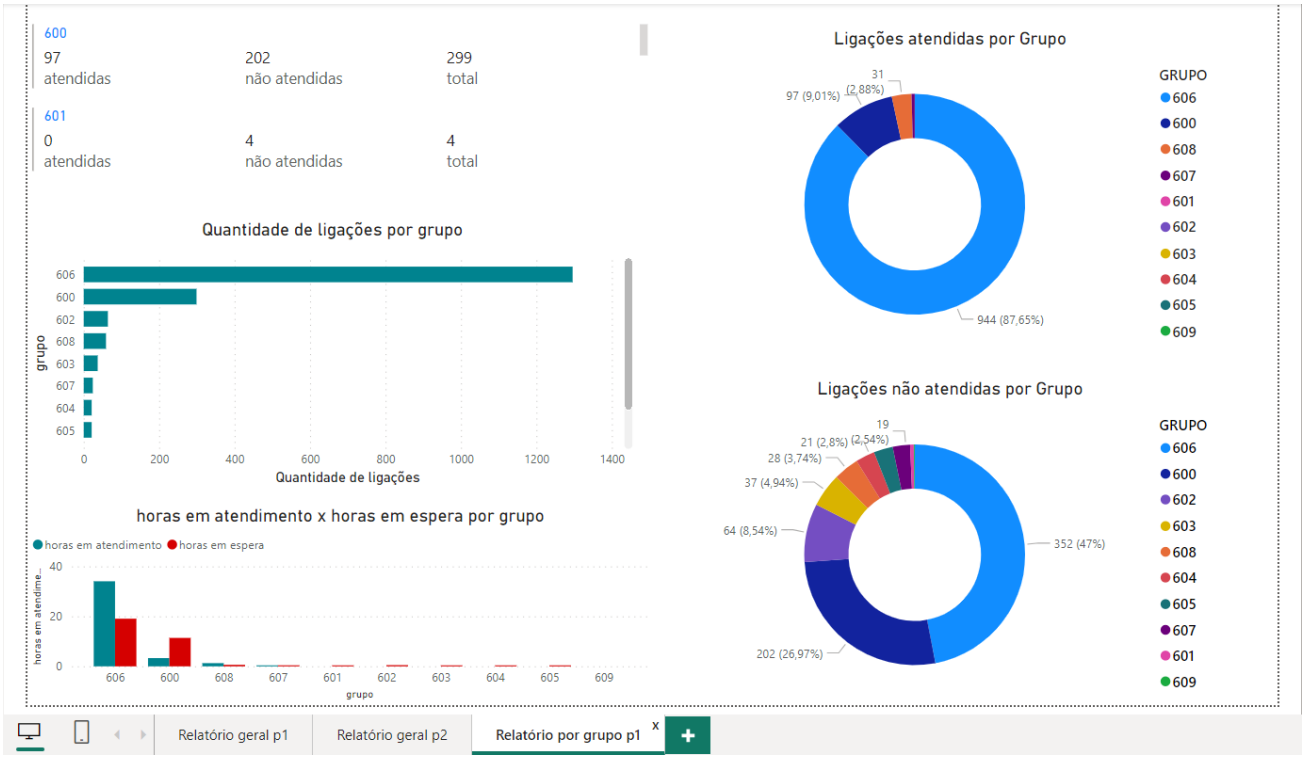
Figura 15 - *Dashboard* geral 2 destacado



Fonte: Autoria própria

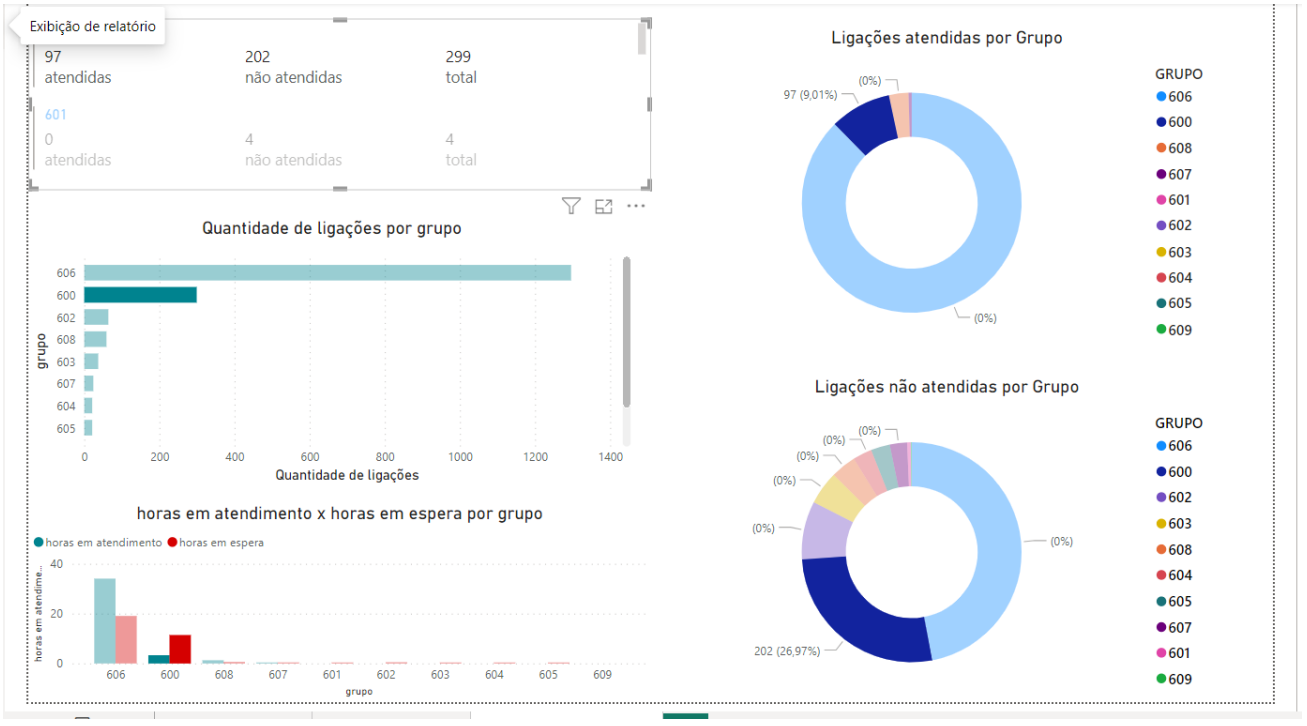
A terceira e última página disponibiliza os dados agrupados pelos grupos. É muito importante ter acesso a esses dados para ter conhecimento de como está o funcionamento de cada grupo, já que muitas vezes os dados quando tratados de forma geral podem esconder o fato de que um grupo pode estar operando da forma esperada e outro pode estar com problemas a serem resolvidos. Seguindo a ideia de ter um *dashboard* interativo, o terceiro *dashboard* possui um cartão contendo todos os grupos que estão na tabela grupos, com a quantidade de chamadas atendidas, não atendidas e o total de recebidas por cada grupo. Esse cartão permite que o usuário selecione um grupo e interaja como todo o *dashboard*, dando um destaque maior para as informações do grupo selecionado. As figuras 16 e 17 mostram, respectivamente, a diferença entre o *dashboard* sem que nenhum grupo específico esteja selecionado e quando o grupo 600 foi selecionado, é possível perceber a diferença na coloração dando destaque às informações do grupo selecionado.

Figura 16 - *Dashboard* de grupos



Fonte: Autoria própria

Figura 27 - *Dashboard* de grupos destacado



Fonte: Autoria própria

Assim como nos dois primeiros *dashboards*, no último também é possível realizar uma análise e obter conhecimento, desta vez acerca de cada grupo especificamente, como por exemplo:

- É possível ver que alguns grupos são praticamente inoperantes, pode ser importante caso a empresa queira saber o porquê isso acontece e como melhorar.
- É possível ver que o grupo 600 atende menos da metade das ligações registradas, essa informação é útil na necessidade de implantar melhorias.

Por fim, após o desenvolvimento dos *dashboards*, o relatório completo com todos os *dashboards* podem ser disponibilizados no *Workspace* do *Power BI*, onde é possível que os usuários autorizados acessem os relatórios e interajam com o mesmo. A disponibilização dos relatórios no *WorkSpace* é interessante pois permite a interação completo do usuário final com os relatórios, permite também que os relatórios sejam atualizados sem que os relatórios antigos sejam perdidos, podendo ser atualizados mensalmente, semanalmente, isso dependerá dos objetivos de negócios de cada empresa. Para o usuário basta ter a licença do *Power BI Pro* para que o mesmo possa acessar os relatórios.

7. Considerações Finais

O objetivo deste trabalho foi destacar a importância que um sistema de *BI* pode ter na gerência de empresas, e mesmo possuindo um foco em telecomunicações, acredito que foi possível identificar que em qualquer departamento ou tipo de negócio é possível notar a melhora na tomada de decisão utilizando a análise de dados.

Além disso, o trabalho também mostrou que é possível utilizar técnicas de programação para desenvolver programas que auxiliam no desenvolvimento de algumas etapas que constituem o sistema de *BI*. Algo que é muito interessante, pois agiliza a extração, tratamento e carga dos dados e também por minimizar os erros.

Por fim, confio que tenha sido possível demonstrar a importância da ciência de dados, como *Business Intelligence*, para potencializar o crescimento de empresas e o alcance aos seus objetivos de negócios.

Para a continuidade deste trabalho, sugere o seguinte trabalho futuro:

- Implementar o sistema de *BI* em tempo real e analisar seu impacto positivo em comparação com o sistema baseado em análises mensais que foi implementado no trabalho.

8. Referências

Bugnion, Pascal; Manivannan, Arun; Nicolas, Patrick R. (2017). **Scala: Guide for Data Science Professionals**. Birmingham: Packt Publishing, 2017.

CARAVANTES, Geraldo Ronchetti ; CARAVANTES, Claudia Born ; KLOECKNER, Monica Caravantes, ***Gestão estratégica de Resultados - Construindo o Futuro***. 1ª ed. v. 1. Porto Alegre: Editora AGE, 2004.

Economist, The (2017). **The world's most valuable resource is no longer oil, but data (2017)**. https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data (2018-07-28).

Rautenberg, Sandro; Carmo, Paulo Ricardo Viviurka do. Big Data e Ciência de Dados: complementariedade conceitual no processo de tomada de decisão. **Brazilian Journal of Information Studies: Research Trends**. 13:1 (2019) p.56-p.67. ISSN 1981-1640.

Eiica (2019). **X Encontro Internacional de Informação, Conhecimento e Ação. Marília**, 2018. http://enancib.marilia.unesp.br/index.php/EIICA/XEIICA. (2019-02-27).

Gartner (2018). **Data Scientist – Gartner IT Glossary** (2018c). https://www.gartner.com/it-glossary/data-scientist (2018-07-28).

GIL, Antônio Carlos **Como Elaborar Projetos de Pesquisa**.. 6. ed. São Paulo: Editora Atlas Ltda., 2017.

APÊNDICE A - CÓDIGO *CONNECT.PY*

```
import pyodbc
import os
from dotenv import load_dotenv

load_dotenv()

user = os.getenv('USER')
password = os.getenv('PASSWORD')
server_oltp = os.getenv('SERVER_OLTP')
server_olap = os.getenv('SERVER_OLAP')
db_oltp = os.getenv('DB_OLTP')
db_oltp_calls = os.getenv('DB_OLTP_CALLS')
db_olap = os.getenv('DB_OLAP')
port = os.getenv('PORT')

# conexão com o banco de dados transacional
connstring_OLTP = f"""DRIVER={{MySQL ODBC 8.0 Unicode Driver}};
                    User ID={user};
                    Password={password};
                    Server={server_oltp};
                    Database={db_oltp};
                    Port={port};
                    String Types=Unicode;"""
```

```
connstring_OLTP_Calls = f"""DRIVER={{MySQL ODBC 8.0 Unicode Driver}};
                User ID={user};
                Password={password};
                Server={server_oltp};
                Database={db_oltp_calls};
                Port={port};
                String Types=Unicode;"""

conn_OLTP = pyodbc.connect(connstring_OLTP)

conn_OLTPCalls = pyodbc.connect(connstring_OLTP_Calls)

#conexão com o banco de dados analítico
connstring_OLAP = f"""DRIVER={{ODBC Driver 17 for SQL Server}};
                Server={server_olap};
                Database={db_olap};
                Trusted_Connection=yes;"""

conn_OLAP = pyodbc.connect(connstring_OLAP)

# Criar o objeto cursor
cursor_OLTP = conn_OLTP.cursor()
cursor_OLTPCalls = conn_OLTPCalls.cursor()
cursor_OLAP = conn_OLAP.cursor()
```

APÊNDICE B - MÓDELO *.ENV.EXAMPLE*

```
SERVER_OLTP =
SERVER_OLAP =
DB_OLTP =
DB_OLTP_CALLS =
DB_OLAP =
PASSWORD =
PORT =
USER =
```

APÊNDICE C - CÓDIGO *SELECTIONS.PY*

```
import config.connect as connect

def selectOLTP(query, db):
    if db == "asterisk":
        connect.cursor_OLTP.execute(query)
        row = connect.cursor_OLTP.fetchall()
    else:
        connect.cursor_OLTPCalls.execute(query)
        row = connect.cursor_OLTPCalls.fetchall()
    return row

def selectOLAP(query):
    connect.cursor_OLAP.execute(query)
    row = connect.cursor_OLAP.fetchall()

    return row
```

APÊNDICE D - CÓDIGO *INSERT.PY*

```
import srcETL.data_transform as transform
import config.connect as connect


def insertGroup(row):
    for i in range(len(row)):
        grpnum = row[i][0]
        strategy = row[i][1]
        grplist = row[i][2]
        name = row[i][3]
                                query_insert = """INSERT INTO grupos
(grpNum,estrategia,grpLista,nome)
                        VALUES (?,?,?,?)"""


            connect.cursor_OLAP.execute(query_insert, (grpnum, strategy, grplist,
name))
        connect.cursor_OLAP.commit()


def insertCalls(rowCall, rowGroup):
    for i in range(len(rowCall)):
        # variáveis vindas da tabela de ligações
        calldate = rowCall[i][0]
        date = transform.date(calldate)
        hours = transform.hours(calldate)
        src = rowCall[i][1]
        dst = rowCall[i][2]
```

```
        dcontext = rowCall[i][3]
        duration = rowCall[i][4]
        disposition = rowCall[i][5]
          # variável criada para armazenar o grupo de cada ramal que recebeu cada ligação
        dstgroup = None
        for j in range(len(rowGroup)):
            # variáveis vindas da tabela dos grupos
            grpnum = rowGroup[j][0]
            grplist = rowGroup[j][2]
                grplist = grplist.split(',') # retorna uma lista com cada elemento da coluna que é separado por vírgula
            for k in range(len(grplist)):
                 if dst == grplist[k] or dst == grpnum: # lista de ramais pertencentes ao grupo
                  dstgroup = grpnum # número do grupo


         query_insert =  """INSERT INTO chamadas (data, hora, origem, destino, contexto, duracao, estado,grpDestino)
                            VALUES (?,?,?,?,?,?,?,?)"""
                connect.cursor_OLAP.execute(query_insert, (date, hours, src, dst, dcontext, duration, disposition, dstgroup))
        connect.cursor_OLAP.commit()
```

APÊNDICE E - CÓDIGO *DATA_TRANSFORM.PY*

```
import config.connect as connect
from datetime import datetime, timedelta
import calendar
from collections import Counter

def date(calldate):
    date = calldate.date()
    date_form = date.strftime('%d-%m-%y')

    return date_form

def hours(calldate):
    hours = calldate.time()

    return hours

def minutesToHours(minutes):
    value = minutes/3600
    hours = int(value)  # Parte inteira representa as horas
    minutes = (value - hours) * 60  # A parte decimal representa os minutos
    minutes = minutes/100
    hours = hours + minutes
    return round(hours,2)
```

```
def callsByGroup(rowGroup, dstlist):
    for i in range(len(rowGroup)):
        count = 0
        answered = 0
        failed = 0
        grpnum = rowGroup[i][0]
        for j in range(len(dstlist)):
            grpid = dstlist[j][0]
            status = dstlist[j][1]
            if grpnum == grpid:
                count = count + 1
                if status == 'ANSWERED':
                    answered = answered + 1
                else:
                    if status == 'NO ANSWER' or status == 'FAILED':
                        failed = failed + 1


        query_insert =  """INSERT INTO chamadasPorGrupo (grpID, quantidade,
atendidas, naoAtendidas)
                           VALUES (?,?,?,?)"""
         connect.cursor_OLAP.execute(query_insert, (grpnum, count, answered,
failed))
        connect.cursor_OLAP.commit()


def hoursByGroup(rowId, rowHours):
    for i in range(len(rowId)):
        grp = rowId[i][0]
        wait = 0
        talk = 0
        for i in range(len(rowHours)):
            status = rowHours[i][1]
            duration = int(rowHours[i][0])
```

```
            dstgroup = rowHours[i][2]
            if grp == dstgroup:
                if status == 'ANSWERED':
                    talk = talk + duration
                else:
                    wait = wait + duration


        mWait = wait/60
        mTalk = talk/60
        hWait = minutesToHours(wait)
        hTalk = minutesToHours(talk)
                    query_insert =  """INSERT INTO tempoPorGrupo (grpID, minutosAtendimento, minutosEspera, horasAtendimento, horasEspera)
                        VALUES (?,?,?,?,?)"""
          connect.cursor_OLAP.execute(query_insert, (grp, mTalk, mWait, hTalk, hWait))
        connect.cursor_OLAP.commit()


def callsByHour(rowDate):


    row = str(rowDate[0][0]).split('-')
    year = int(row[0])
    month = int(row[1])
    day = int(row[2])
    maxday = calendar.monthrange(year,month)[1]
    hour = 0
    while day <= maxday:
        date = datetime(year, month, day)
        initialHour = datetime(year, month, day, hour)
        interval = timedelta(hours = 1)
        query_verif = """SELECT * FROM chamadas WHERE data = ?"""
        connect.cursor_OLAP.execute(query_verif, (date))
        verif = connect.cursor_OLAP.fetchall()
```

```
        if(len(verif) > 0):
            while initialHour < datetime(year, month, day + 1, 0):
                finalHour = initialHour + interval
                    query_select = """SELECT * FROM chamadas WHERE data = ?
AND hora >= ? AND hora <= ?"""
                                    connect.cursor_OLAP.execute(query_select,  (date,
initialHour.time(), finalHour.time()))
                row = connect.cursor_OLAP.fetchall()
                number = len(row)


                if (number > 0):
                        query_insert = """INSERT INTO chamadasPorHora (data, hora,
quantidade)
                            VALUES (?,?,?)"""
                                    connect.cursor_OLAP.execute(query_insert,  (date,
initialHour.time(), number))
                    connect.cursor_OLAP.commit()


                initialHour = initialHour + interval
            day = day + 1
        else :
            day = day + 1


def createCalendar(rowDate):


    months_pt = [
        "Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",
        "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"
    ]


    for i in range(len(rowDate)):
        date = rowDate[i][0]
        row = str(rowDate[i][0]).split('-')
```

```
        year = int(row[0])
        month = int(row[1])
        monthName = months_pt[month - 1]
        day = int(row[2])


         query_insert = """INSERT INTO calendario (data, ano, mes, mesNome,
dia) VALUES (?,?,?,?,?)"""
                connect.cursor_OLAP.execute(query_insert, (date, year, month,
monthName, day))


def createMeasures(rowDate, rowHours, rowDst, rowGroup):


     avgCalls, avgWait, avgTalk, mTalk, hTalk, mWait, hWait, answered, failed,
total = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    strategy = ''
    avgCalls = average(rowDate)
        mTalk, mWait, hTalk, hWait, avgTalk, avgWait = totalHours(rowHours,
rowDate)
    answered, failed, total = totalCalls(rowDst)
    strategy = maxStrategy(rowGroup)
          query_insert =  """INSERT INTO medidas (minutosAtendimento,
minutosEspera, horasAtendimento, horasEspera, mediaLigacoes,
                                    tempoMedioAtendimento, tempoMedioEspera,
atendidas, naoAtendidas, total, estrategiaComum)
                        VALUES (?,?,?,?,?,?,?,?,?,?,?)"""
      connect.cursor_OLAP.execute(query_insert, (mTalk, mWait, hTalk, hWait,
avgCalls, avgTalk, avgWait, answered, failed, total, strategy))
    connect.cursor_OLAP.commit()



def average(rowDate):


    count = len(rowDate)
    date = str(rowDate[-1][0]).split('-') # Seleciona a última data registrada
```

```
    day = int(date[2])

    mediaCalls = count/day

    return mediaCalls

def totalHours(rowHours, rowDate):
    count = len(rowDate)
    wait = 0
    talk = 0
    for i in range(len(rowHours)):
        status = rowHours[i][1]
        duration = int(rowHours[i][0])
        if status == 'ANSWERED':
            talk = talk + duration
        else:
            wait = wait + duration

    minutesWait = wait/60
    minutesTalk = talk/60
    hoursWait = minutesToHours(wait)
    hoursTalk = minutesToHours(talk)
    mediaTalk = minutesTalk/count
    mediaWait = minutesWait/count
    hoursWait = round(hoursWait,2)
    hoursTalk = round(hoursTalk,2)
    return minutesTalk, minutesWait, hoursTalk, hoursWait, round(mediaTalk,2),
round(mediaWait,2)

def totalCalls(rowDst):
    answered = 0
```

```
    failed = 0
    total = 0
    for i in range(len(rowDst)):
        total = total + 1
        status = rowDst[i][1]
        if status == "ANSWERED":
            answered = answered + 1
        else:
            if status == 'NO ANSWER' or status == 'FAILED':
                failed = failed + 1

    return answered, failed, total

def maxStrategy(rowGroup):
    strategyList = []
    for i in range(len(rowGroup)):
        strategy = rowGroup[i][1]
        strategyList.append(strategy)
    count = Counter(strategyList)
    value = max(count, key=count.get)
    return value
```

APÊNDICE F - CÓDIGO *MAIN.PY*

```
import srcETL.data_transform as transform

import srcETL.selections as select

import srcETL.insertions as insert


# SELECT para a tabela dGrupos

rowGroup = select.selectOLTP("SELECT grpnum,strategy,grplist,description FROM ringgroups",

                             "asterisk")

insert.insertGroup(rowGroup)


# SELECT para a tabela fChamadas

rowCall = select.selectOLTP("""SELECT calldate, src, dst, dcontext, duration, disposition

                               FROM cdr WHERE calldate BETWEEN '2023-07-01' AND '2023-07-31'""",

                            "asteriskcdrdb")

insert.insertCalls(rowCall, rowGroup)


# SELECT's para a tabela dChamadasPorGrupo

rowId = select.selectOLAP("SELECT grpNum FROM grupos")

rowDst = select.selectOLAP("SELECT grpDestino, estado FROM chamadas")

transform.callsByGroup(rowId, rowDst)


# SELECT para a tabela dHorasPorGrupo

rowHours = select.selectOLAP("SELECT duracao, estado, grpDestino FROM chamadas")
```

```
transform.hoursByGroup(rowId, rowHours)

# SELECT para a tabela dChamadasPorGrupo
rowDate = select.selectOLAP("SELECT data FROM chamadas")
transform.callsByHour(rowDate)

# Chamada para a construção da tabela dCalendar
rowDateDistinct = select.selectOLAP("SELECT DISTINCT data FROM chamadas ORDER BY data")
transform.createCalendar(rowDateDistinct)

# Chamada para as funções totalHours que irão compor a tabela de medidas
transform.createMedidas(rowDate, rowHours, rowDst, rowGroup)
```