UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Fernandes Vieira

Desenvolvimento de uma ferramenta para visualização de cubos de dados gerados a partir de consultas SQL com agrupamentos

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Fernandes Vieira

Desenvolvimento de uma ferramenta para visualização de cubos de dados gerados a partir de consultas SQL com agrupamentos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Profº Drº Humberto Luiz Razente

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2024

Gabriel Fernandes Vieira

Desenvolvimento de uma ferramenta para visualização de cubos de dados gerados a partir de consultas SQL com agrupamentos

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 0 de maio de 2024:

Profº Drº Humberto Luiz Razente
Orientador

Professor

Professor

Uberlândia, Brasil
2024

Resumo

Este trabalho aborda o desenvolvimento de um sistema web que permite a visualização de dados a partir de consultas SQL utilizando agrupamento de dados GROUP BY CUBE. O sistema oferece uma interface intuitiva onde os usuários podem inserir consultas SQL e enviá-las para análise. Por meio de uma conexão estabelecida com o banco de dados através do PHP, o sistema recebe os resultados da consulta e os converte para o formato JSON. Em seguida, esses dados são utilizados para construir uma tabela dinâmica que possibilita a plotagem das dimensões (linha, coluna e profundidade), proporcionando uma análise dos dados. O trabalho inclui a realização de testes para avaliar a funcionalidade do sistema, analisando os resultados obtidos e verificando sua conformidade com os objetivos estabelecidos. Este sistema representa uma contribuição significativa para a área de visualização de dados, sendo disponibilizado com código livre, oferecendo uma ferramenta para análise de dados tridimensionais de forma acessível e eficiente.

Palavras-chave: OLAP, Sistema, GROUP BY CUBE, Banco de Dados.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

OLAP	*Online Analytical Processing*
JSON	*JavaScript Object Notation*
PHP	*Hypertext Preprocessor*
CSS	*Cascading Style Sheets*
HTML	*HyperText Markup Language*
JS	*JavaScript*
CSV	*Comma-separated values*
CPI	Índice de Preços ao Consumidor
OCI	*Oracle Call Interface*

Sumário

1 Introdução

A análise estratégica de uma organização permite que os usuários de tomada de decisão identifiquem tendências e padrões para melhor conduzir os negócios de sua empresa. A utilizaçao de recursos que facilitem esta análise é indispensável.

O crescente poder de processamento e sofisticação das ferramentas e técnicas analíticas levou ao desenvolvimento dos chamados *data warehouses*. Esses *data warehouses* fornecem armazenamento, funcionalidade e capacidade de resposta a consultas além das capacidades dos bancos de dados orientados a transações (ELMASRI; NAVATHE, 2014).

Elmasri e Navathe (2014) define OLAP (processamento analítico on-line) como um termo usado para descrever a análise de dados multidimencionais. As consultas OLAP tem como principal objetivo concentrar-se nos dados que geralmente são utilizados pelo gerenciamento no processo de tomada de decisão. Portanto, dados que mostram classificação, tendências, subtotais, totais gerais, etc, são informações de grande interesse para a administração de uma empresa. As consultas OLAP recuperam dados de um data Warehouse (TANIAR; RAHAYU, 2022).

De acordo com Sherman (2014), a evolução contínua do ambiente empresarial, caracterizado pela crescente complexidade dos dados e das operações, destaca ainda mais a importância da análise estratégica para a tomada de decisões informadas e eficazes. Nesse cenário dinâmico, as organizações enfrentam desafios para extrair informações significativas de seus dados e utilizá-los de forma a impulsionar o sucesso empresarial.

Dessa forma, com o uso de ferramentas de processamento de dados analíticos (OLAP) e recursos de *Data Warehouse*, a necessidade de desenvolver um sistema para auxiliar na análise estratégica de dados organizacionais representados visualmente por cubos de dados OLAP constitui a principal motivação para o desenvolvimento deste trabalho.

Para ajudar na visão dos dados de uma empresa, neste trabalho será elaborado um protótipo onde, a partir da consulta SQL, utilizando agrupamento de dados *GROUP BY CUBE*, pode-se visualizar os resultados no sistema.

1.1 Justificativa

A importância da análise de dados tem aumentado constantemente desde o início da década de 1990, pois as organizações de todos os setores estão sendo obrigadas a melhorar seus processos de tomada de decisão para manter sua vantagem competitiva (VAISMAN; ZIMÁNYI, 2014).

Portanto, ferramentas e recursos como banco de dados relacionais e o processamento de dados analíticos (OLAP) surgem para auxiliar nessa investigação dos dados. O banco de dados é utilizado para armazenar informações e o OLAP para recuperá-las, ambos são especializados para exercer suas funções de forma eficiente (SUMATHI; SIVANANDAM, 2006).

Dentre os desafios encontrados, os modelos de dados OLAP tendem a ser multidimensionais. Os resultados encontrados não estão em um formato apropriado para apresentação. Esses dados são considerados "brutos". Dessa forma, os dados recuperados precisam ser formatados adequadamente para serem apresentados (TANIAR; RAHAYU, 2022).

Portanto, torna-se importante o desenvolvimento de uma ferramenta que possibilite o mapeamento e a visualização dos dados em um modelo multidimensional.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver uma ferramenta web no qual a partir de uma consulta SQL, com agrupamento de dados *GROUP BY CUBE*, seja possível a visualização dos resultados tridimensionais do sistema, possibilitando também a escolha de quais dimensões serão plotadas no plano principal e qual dimensão será plotada na profundidade. Os objetivos específicos deste trabalho são:

- Realizar o planejamento e análise do problema para realizar o desenvolvimento do software.
- Preparar o ambiente com o banco de dados e as ferramentas que serão utilizadas durante a implementação.
- Desenvolver o sistema que atenda às necessidades encontradas.
- Realizar testes para demonstrar a funcionabilidade do protótipo com um conjunto de dados reais.

1.3 Metodologia

Esta seção descreve a abordagem metodológica adotada para a realização deste trabalho, destacando os passos e procedimentos utilizados para alcançar os objetivos propostos.

Inicialmente, foi realizado um estudo da literatura sobre os principais temas envolvidos no projeto. Isso incluiu uma revisão abrangente sobre banco de dados relacional,

técnicas de agrupamento de dados e o conceito de *GROUP BY CUBE*. Essa revisão proporcionou uma compreensão aprofundada dos fundamentos teóricos e práticos necessários para o desenvolvimento do sistema proposto.

Para a implementação do sistema, foram utilizadas diversas linguagens e tecnologias. O *back-end* do sistema foi desenvolvido utilizando a linguagem de programação PHP, enquanto o *front-end* foi construído utilizando JavaScript, HTML e CSS. Essa abordagem permite uma separação entre a lógica e a apresentação dos dados.

Como base de dados, optou-se por utilizar o banco de dados Oracle, complementado pelo uso do PostgreSQL. A gestão do banco de dados foi realizada utilizando a ferramenta SQL *Developer* e, além disso, o servidor Xampp foi empregado para facilitar a conexão entre o banco de dados Oracle e o ambiente de desenvolvimento PHP.

Para validar a funcionalidade do sistema desenvolvido, foi feito um estudo de caso utilizando um conjunto de dados reais de uma rede de varejo. A base de dados permite a computação de análises tridimensionais, em formato de cubos de dados.

2 Referencial Teórico

Este capítulo apresenta um levantamento bibliográfico acerca dos fundamentos teóricos envolvidos na realização desta monografia e para o entendimento do trabalho.

2.1 Banco de Dados Relacional

Bancos de dados relacionais, de acordo com Vaisman e Zimányi (2014), têm sido usados para armazenar informações em muitos domínios de aplicação. Apesar das alternativas tecnologicas de banco de dados que surgiram nas últimas décadas, o modelo relacional ainda é a abordagem mais usada para armazenar informações persistentes e cruciais para a operação diária de uma organização.

Os modelos relacionais são baseados no conceito de relação matemática, que pode ser vista como uma tabela de valores, na teoria dos conjuntos e na lógica de predicados de primeira ordem. O cálculo relacional é considerado a base da linguagem SQL, e a álgebra relacional é usada internamente nas implementações de banco de dados para processamento e otimização de consultas (ELMASRI; NAVATHE, 2014).

O modelo possui uma estrutura de dados simples, uma relação (tabela) composta por um ou vários atributos (colunas). Assim, um esquema relacional descreve a estrutura de um conjunto de relações. Informalmente, cada relacionamento é como uma tabela de valores ou, até certo ponto, um arquivo simples de registros (VAISMAN; ZIMÁNYI, 2014).

Em termos formais, uma linha é chamada de tupla, o cabeçalho da coluna é chamado de atributo. Um tipo de dados que descreve o tipo de valores que podem aparecer em cada coluna é representado por um campo de valores possíveis. Cada linha na tabela representa uma coleção de valores de dados relacionados. Uma linha representa um fato que normalmente corresponde a uma entidade ou relacionamento do mundo real. Os nomes da tabela e de coluna são usados para ajudar a interpretar o significado dos valores em cada linha (ELMASRI; NAVATHE, 2014).

Álgebra relacional é uma coleção de operações para manipular relações. Essas operações podem ser unárias, tomando uma relação como argumento e retornando outra relação, ou binárias, tomando duas relações como argumentos e retornando uma relação. A álgebra é fechada e as operações podem ser combinadas para calcular as respostas de algumas consultas. Além disso, outras classificações das operações são: operações básicas, que não podem ser derivadas da combinação de outras operações, e as operações derivadas, que são uma sequência de cálculos básicos, definidas para tornar as consultas mais fáceis de expressar (ELMASRI; NAVATHE, 2014).

2.1.1 Dados Relacionais e consultas SQL

SQL (linguagem de consulta estruturada) é a linguagem mais comum para criação, manipulação e recuperação de dados de banco de dados relacionais (VAISMAN; ZIMÁNYI, 2014).

Os sistemas relacionais modelam dados N-dimensionais como uma relação com domínios de N-atributos. Por exemplo, uma tabela de clima (Tabela 1) representa os dados de temperatura da terra em 4 dimensões, as primeiras quatro colunas: latitude, longitude, altitude e tempo. Colunas adicionais podem representar medições nos pontos 4D, como temperatura, pressão, umidade e velocidade do vento. Cada medição meteorológica individual é registrada como uma nova linha desta tabela. Esses valores medidos podem ser agregados ao longo do tempo (hora) ou no espaço (uma área de medição centralizada no ponto)(GRAY et al., 1997).

CLIMA					
TEMPO	LATITUDE	LONGITUDE	ALTITUDE(M)	TEMP.(C)	PRESSÃO(MB)
96/6/1:1500	37:58:33N	122:45:28W	102	21	1009
...					
96/6/7:1500	34:16:18N	27:05:55W	10	23	1024

Tabela 1 – Tabela de Clima - Fonte: Gray et al. (1997)

As ferramentas de visualização e análise de dados fazem uso extensivo da redução de dimensionalidade (agregação) para obter uma melhor compreensão. Normalmente, os dados de outras dimensões não incluídos na representação 2D são agregados na forma de um histograma, tabelas de contingência, subtotais etc. No padrão SQL, dependemos de funções agregadas e do operador GROUP BY para apoiar a agregação(GRAY et al., 1997).

As funções de agregação do SQL são amplamente utilizadas em aplicativos de banco de dados. O padrão SQL (ISO/IEC 9075:1992 International Standard for Database Language SQL) (ISO, 1992) fornece cinco funções para agregar os valores em uma tabela: COUNT(), SUM(), MIN(), MAX() e AVG( ). Recentemente, foram adicionadas mais duas funções sendo elas STDDEV() e VARIANCE(). Por exemplo, a média de todas as temperaturas medidas é expressa como:

SELECT AVG(Temp) FROM Clima

Funções agregadas retornam um único valor. Usando a construção GROUP BY, o SQL pode criar uma tabela de muitos valores agregados com base nos valores distintos do conjunto de atributos de agrupamento. Por exemplo, ele particiona a relação em conjuntos de tuplas disjuntos e, em seguida, faz a agregação em cada conjunto (GRAY et al., 1997).

A consulta a seguir informa a temperatura média em relação a cada tempo e altitude dos dados coletados:

SELECT Tempo, Altitude ,AVG(Temp)

FROM Clima GROUP BY Tempo, Altitude

Um problema no uso do GROUP BY refere-se a consultas usando totais e subtotais para relatórios de detalhamento. As informações geralmente são agregadas em um nível de alta granularidade e, em seguida, em níveis sucessivamente mais refinados (GRAY et al., 1997).

2.1.2 Operadores CUBE e ROLLUP

O operador do cubo de dados cria uma tabela contendo todos os valores agregados. O total agregado usando a função f() é representado como a tupla (GRAY et al., 1997):

ALL, ALL, ALL, ... , ALL, f(*)

Para a criação de um cubo de dados é necessário a geração do conjunto de todos os subconjuntos das colunas de agregação. Como o CUBE é uma operação de agregação, utilizamos junto ao operador GROUP BY, sendo ele um operador relacional, com CUBE e ROLLUP como formas alteradas do operador (GRAY et al., 1997). A Figura 1 apresenta um exemplo da sintaxe de um cubo.

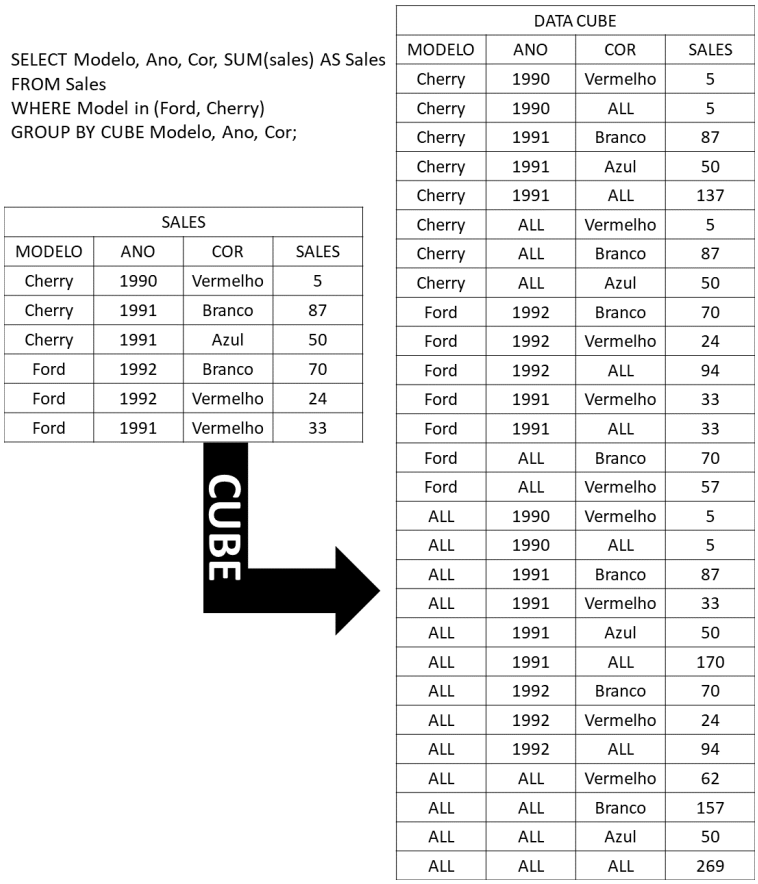


Figura 1 – Cubo de dados 3D construído da tabela à esquerda pela instrução CUBE - Fonte: Adaptado de Gray et al. (1997)

Como outro exemplo, segue uma declaração para agregar o conjunto de observações de temperatura, levando em conta a Tabela 1:

SELECT dia, pais, MAX(Temp)

FROM Clima

GROUP BY CUBE Dia(Tempo) AS dia,

Pais(Latitude, Longitude)AS pais;

A semântica do operador CUBE é que ele primeiro agrega todos os atributos na cláusula GROUP BY e, em seguida, ele faz UNIONs em cada superagregação do cubo global substituindo ALL pelas colunas de agregação (GRAY et al., 1997).

O operador SQL ROLLUP é útil para criar relatórios mais enxutos e focados, especialmente quando um cubo de dados completo seria excessivo. Por exemplo, um atributo de data que pode ser analisado por semana, mês e ano. Embora consultas que agregam por ano, semana ou dia sejam comuns, criar um cubo que combine todas essas dimensões pode não ser necessário e até redundante.

Em vez disso, o ROLLUP permite que você produza apenas os superagregados, ou seja, as agregações cumulativas que fazem mais sentido:

(v1 ,v2 ,...,vn, f()),

(v1 ,v2 ,...,ALL, f()),

...

(v1 ,ALL,...,ALL, f()),

(ALL,ALL,...,ALL, f()).

Isso permite que você trabalhe com agregados cumulativos, como soma ou média acumulada, de forma mais eficiente e natural. Enquanto o cubo completo de dados pode ser complexo e não linear devido ao grande número de combinações possíveis de atributos no GROUP BY, o ROLLUP simplifica essa análise ao fornecer uma sequência lógica de agregados, ideal para relatórios resumidos. ROLLUP e CUBE devem ser solicitados para uma aplicação de operadores cumulativos (GRAY et al., 1997).

2.2 Esquema Estrela e Floco de Neve

Embora estritamente não faça parte do design do operador CUBE e ROLLUP, há um importante conceito de design de banco de dados que facilita o uso de operações de agregação. É comum registrar eventos e atividades com um registro detalhado dando todas as dimensões do evento. Por exemplo, o registro do item de venda na Figura 2 fornece a identificação do comprador, vendedor, produto adquirido, unidades adquiridas,

preço, data e escritório de vendas que está creditado com a venda (GRAY et al., 1997).

O esquema geral da Figura 2 possui nome: esquema floco de neve. Esquemas mais simples que possuem uma única tabela de dimensão para cada dimensão são chamados de esquema em estrela. As consultas nesses esquemas são chamadas de consultas em floco de neve e consultas em estrela, respectivamente (GRAY et al., 1997).

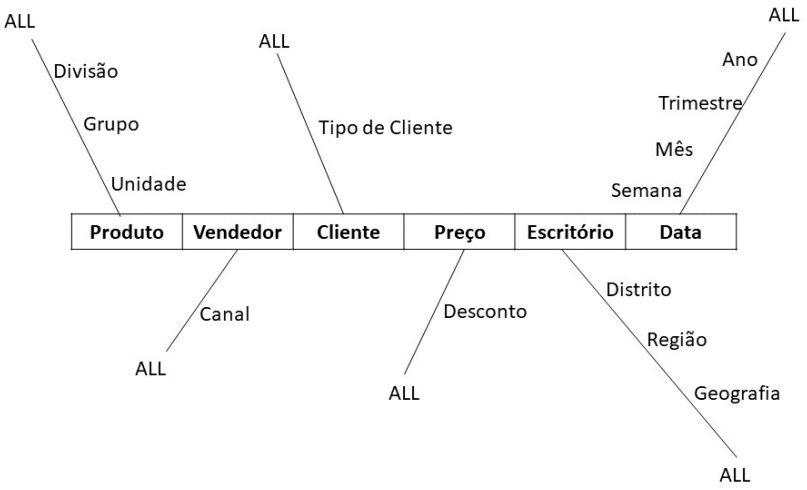


Figura 2 – Um esquema de floco de neve mostrando a tabela de fatos principal e algumas das muitas granularidades de agregação das dimensões principais. Fonte: Adaptado de Gray et al. (1997)

A tabela complementar lista os atributos para cada valor de dimensão. As tabelas de dimensão também podem conter informações que descrevem outras propriedades de uma dimensão. Essas tabelas especificam uma variedade de detalhes de montagem para as dimensões. O usuário pode querer agregar várias dimensões e, em seguida, acumular o cubo em um ou todas as visões dos atributos (TANIAR; RAHAYU, 2022).

O diagrama da Figura 2 mostra que as dimensões formam uma hierarquia e uma rede. Como exemplo, os dias agregam-se em semanas, porém as semanas não se aninham em meses, trimestres ou anos. É possível pensar em datas em termos de dias úteis, fins de semana, feriados. Portanto, um gráfico de dimensões mais completo da Figura 2 seria bastante complexo (GRAY et al., 1997).

2.3 DataWarehouse

Um *datawarehouse* é uma base de dados específica direcionada para o apoio à decisão. Ela recolhe dados de várias bases de dados operacionais e de outras fontes de dados e transforma-os em novas estruturas que se adaptam melhor à tarefa de efetuar análise de negócio (VAISMAN; ZIMÁNYI, 2014). O *datawarehouse* é uma visão multidimensional das bases de dados com agregados e resumos pré-computados. Em muitos aspectos, trata-

se basicamente de fazer agregados antecipadamente, ou seja, a pré-computação feita ao nível da concepção e não ao nível da consulta (TANIAR; RAHAYU, 2022).

Definido anteriormente, um banco de dados é uma coleção de dados relacionados. Um *datawarehouse* também é uma coleção de informações, bem como um sistema de suporte. Contudo, de acordo com Elmasri e Navathe (2014) existe uma distinção clara: "Os bancos de dados tradicionais são transacionais (relacionais, orientados a objeto, em rede ou hierárquicos), os data warehouses têm a característica distintiva de servir principalmente para aplicações de apoio à decisão. Eles são otimizados para recuperação de dados, e não para processamento de transações de rotina".

Rainardi (2008) caracterizou um data warehouse como uma coleção de dados orientada a objeto, integrada, não volátil, variável no tempo para o suporte às decisões da gerência. Estas características podem ser explicadas de acordo com Vaisman e Zimányi (2014):

- Orientado a objeto significa que os data warehouses se centram nas necessidades analíticas de diferentes áreas de uma organização. Estas áreas variam consoante o tipo de atividades realizadas pela organização.
- Integrado significa que os dados foram obtidos de vários sistemas operacionais e externos, o que implica a resolução de problemas devidos a diferenças na definição e conteúdo dos dados, tais como diferenças de formato e codificação dos dados, sinônimos, homônimos, multiplicidade de ocorrências de dados e muitos outros.
- Não volátil significa que a durabilidade dos dados é assegurada pelo fato de não permitir a sua modificação e remoção de dados, alargando assim o âmbito dos dados a um período de tempo mais longo do que os sistemas operacionais normalmente oferecem.
- A variação no tempo indica a possibilidade de reter valores diferentes para a mesma informação, bem como o momento em que ocorreram alterações desses valores. Por exemplo, uma base de dados num banco pode armazenar informação sobre o saldo médio mensal das contas dos clientes durante um período de vários anos.

A Figura 3 fornece uma visão geral da estrutura do conceito de *data warehouse*. Ele demonstra todo o processo de *datawarehouse*, incluindo a capacidade de limpar e reformatar dados antes de carregá-los no *datawarehouse*. Esse processo é tratado por ferramentas conhecidas como ferramentas ETL (*Extract, Transform, and Load*) (ELMASRI; NAVATHE, 2014).

Um *data warehouse* também suporta vários níveis de agregação sob a forma de uma hierarquia de data warehouses de diferentes granularidades. Isto resulta numa ar-

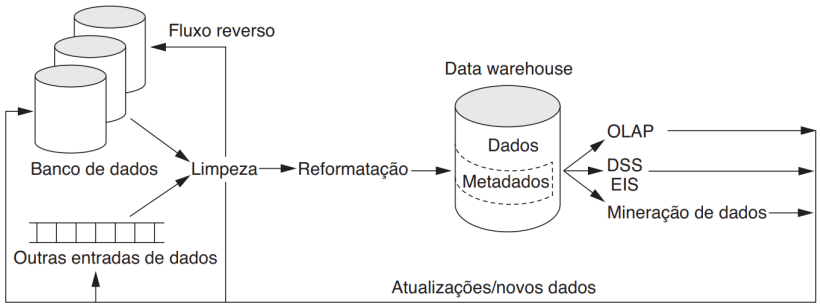


Figura 3 – Exemplo de transações: Data Warehouse. Fonte: Elmasri e Navathe (2014)

quitetura de *datawarehousing*. Com a disponibilidade de *datawarehouses* em diferentes granularidades, é possível consultar o armazém de dados de diferentes níveis, consoante a granularidade da análise para a tomada de decisões (TANIAR; RAHAYU, 2022).

2.4 Processamento de Dados Analíticos (OLAP)

Um OLAP é uma consulta SQL que recupera dados do banco de dados. As consultas se concentram em dados que geralmente seriam usados, por exemplo, pela administração de uma empresa para a tomada de determinadas decisões. Portanto, dados que mostram classificações, tendências, subtotais, totais gerais, etc, seriam de grande interesse para essas pessoas (TANIAR; RAHAYU, 2022).

Porém, os resultados recuperados de uma consulta não estão em um formato apropriado para uma apresentação. Os dados recuperados precisam ser devidamente formatados e apresentados usando ferramentas, que geralmente incluem gráficos. Portanto, o papel do OLAP é recuperar os dados necessários do banco de dados e apresentar esses dados "brutos" ao *Business Intelligence* para processamento (VAISMAN; ZIMÁNYI, 2014).

Uma característica fundamental do modelo multidimensional é que permite visualizar dados de múltiplas perspectivas e em vários níveis de detalhe. As operações OLAP permitem essas perspectivas e níveis de detalhe a ser materializado explorando as dimensões e suas hierarquias, fornecendo assim um ambiente interativo de análise de dados.

O exemplo retirado de Vaisman e Zimányi (2014), mostrando o funcionamento de operações OLAP. A Figura 4 apresenta um cenário de vendas trimestrais (em milhares) por categorias (produtos) e cidades (clientes) para o ano de 2012 de uma determinada empresa. O exemplo mostra como um usuário final pode operar sobre um cubo de dados para analisar os dados de maneiras diferentes. O usuário começa da Figura 4a.

A primeira ação desejada é calcular as quantidades de vendas por país. Para isto, aplica-se uma operação de roll-up ao nível do País ao longo da dimensão Cliente. O resultado é apresentado na Figura 4b. Enquanto o cubo original continha quatro valores

na dimensão Cliente, um para cada cidade, o novo cubo contém dois valores, cada um correspondendo a um país. O restante as dimensões não são afetadas.

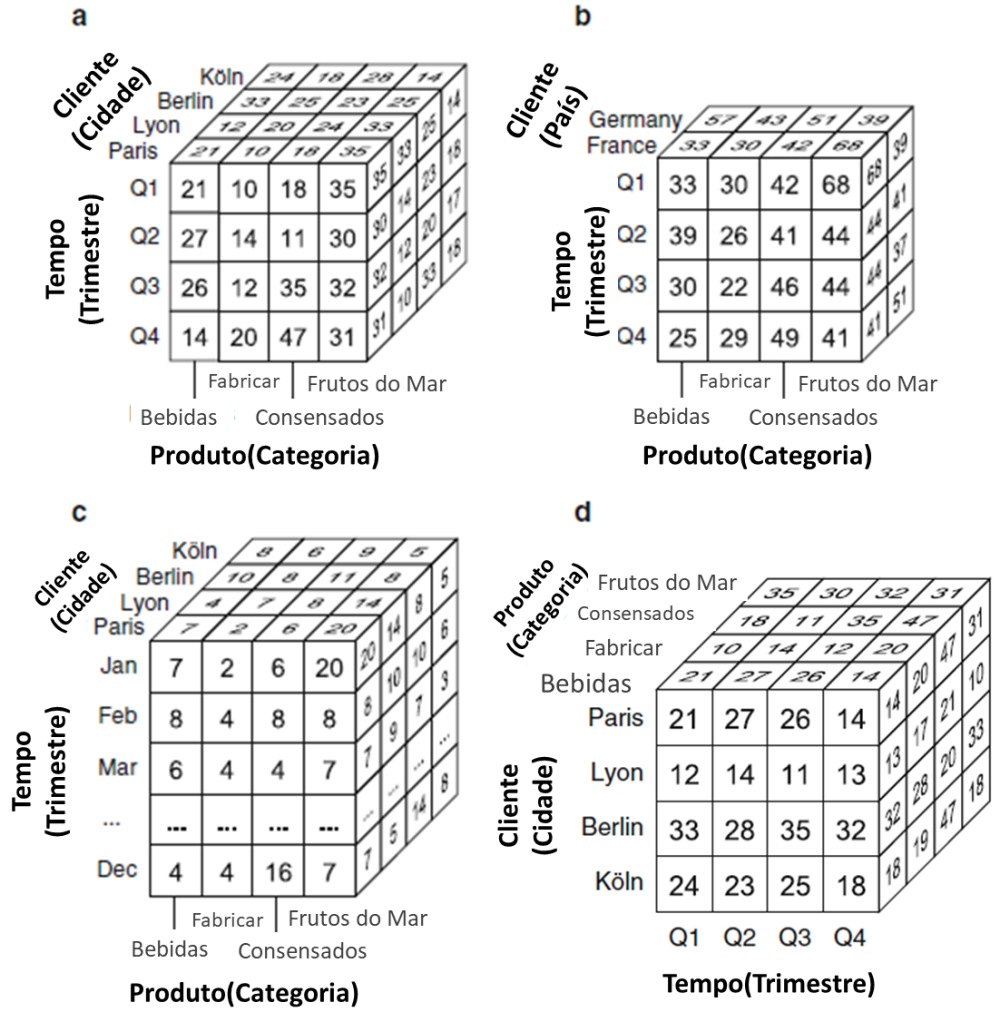


Figura 4 – Operações OLAP. (a) Cubo original. (b) Roll-up da dimensão Cliente para o nível do país. (c) Drill-down da dimensão Tempo para o nível Mês. (d) Pivoteamento. Fonte: Vaisman e Zimányi (2014)

Em seguida, foi notado que as vendas da categoria frutos do mar na França estão significativamente maiores no primeiro trimestre em comparação com os outros. Dessa forma, é feita uma busca detalhada ao longo da dimensão Tempo até o nível Mês para descobrir se esse valor alto ocorreu durante um determinado mês. Assim ela descobre que, por algum motivo, as vendas em Janeiro dispararam tanto em Paris quanto em Lyon, como mostrado na Figura 4c.

Por fim, é feito um pivoteamento onde pegamos o cubo original e giramos os eixos do cubo sem alterar as granularidade, sendo possível a visão do cubo com a dimensão Tempo no eixo x, conforme mostrado na Figura 4d.

Além das operações básicas descritas acima, as ferramentas OLAP fornecem uma grande variedade de operações matemáticas, estatísticas e financeiras para taxas de cál-

culo, variações, juros, depreciação, conversões de moeda, etc.

2.5 JSON

JavaScript Object Notation (JSON) é um formato leve baseado nos tipos de dados da linguagem de programação JavaScript. Destaca-se pela simplicidade e facilidade de leitura e escrita por humanos e máquinas. Sua popularidade decorre do fato de ser um padrão aberto e independente de linguagem, tornando-o ideal para integração entre diferentes tecnologias e rapidamente se tornando um dos formatos mais populares para troca de dados na web (BRAY, 2014).

De acordo com Bourhis, Reutter e Vrgoč (2020) os documentos JSON, na sua essência, são dicionários constituídos por pares chave-valor, em que o valor pode ser novamente um documento JSON, permitindo assim um nível arbitrário de aninhamento. Um exemplo de um documento JSON é apresentado na Figura 5.

```
{
  "nome": "Gabriel",
  "idade": 23,
  "casado": false,
  "interesses": ["tecnologia", "livros"],
  "endereco": {
    "rua": "Rua das Flores",
    "numero": 123,
    "cidade": "Uberlândia"
  }
}
```

Figura 5 – Exemplo documento JSON. Fonte: O Autor

Um documento JSON é composto por uma coleção de pares chave-valor, onde as chaves são strings e os valores podem ser strings, números, booleanos, arrays, objetos ou null. A estrutura básica é um objeto sendo delimitada por chaves . Cada par chave-valor é separado por vírgula, e as chaves são seguidas por dois pontos (MCPEAK, 2015).

Algumas das principais características, segundo Bourhis, Reutter e Vrgoč (2020) e McPeak (2015), são:

- Troca de dados: JSON é comumente usado para enviar e receber dados entre clientes e servidores em aplicações web.
- Armazenamento de configurações: Devido à sua estrutura simples, configurações do aplicativo, como preferências do usuário ou configurações do sistema, podem ser armazenadas em arquivos JSON.

- Persistência de dados: O banco de dados usa JSON como formato de armazenamento nativo, permitindo representação flexível de dados sem a necessidade de um esquema fixo.
- Configurando APIs RESTful: Muitas APIs RESTful usam JSON para representar recursos e suas propriedades porque é compatível com JavaScript e fácil de interpretar por clientes web.

Dessa forma, o JSON desempenha um papel fundamental na comunicação entre sistemas distribuídos na web, oferecendo uma maneira simples e eficaz de representar e transmitir dados (BRAY, 2014).

3 Desenvolvimento

O capítulo é dedicado à exposição do processo de concepção, implementação e validação do projeto proposto. Neste contexto, são descritas todas as etapas percorridas desde a definição dos requisitos até a entrega do produto final. Além disso, são apresentadas as decisões tomadas ao longo do desenvolvimento e também uma visão abrangente das tecnologias, metodologias e abordagens adotadas para alcançar os objetivos estabelecidos.

3.1 Ambiente

A implementação do presente projeto foi dividida em duas partes distintas: o *front-end* e o *back-end*. Cada parte foi desenvolvida utilizando tecnologias específicas para garantir a funcionalidade do sistema como um todo.

3.1.1 *Front-end*

Para a construção da interface do usuário, foram empregadas as linguagens HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) e *JavaScript* (JS).

O HTML foi utilizado para estruturar o conteúdo da página, definindo os elementos e sua organização, baseando-se no uso de tags e elementos para estruturar o conteúdo e criar uma página WEB (Mozilla contributors, 2023). Já o CSS foi empregado para estilizar e formatar os elementos, proporcionando uma experiência visual agradável e consistente aos usuários (Mozilla contributors, 2022). Foi utilizado também o Bootstrap, um framework *front-end* que fornece componentes que auxiliam o desenvolvedor a criar designs mais complexos, como carrosséis de imagens, modais, navbar, entre outros (W3S, 2021).

Além disso, o JavaScript foi utilizado para adicionar interatividade à interface do usuário, permitindo a manipulação dinâmica dos elementos da página, validação de formulários e interações com o servidor.

3.1.2 *Back-end*

A camada lógica de negócios é responsável por processar as requisições recebidas do *front-end*, executar a lógica do sistema e interagir com a camada de armazenamento de dados. Para o *back-end* deste projeto, optou-se por utilizar a linguagem de programação PHP (*Hypertext Preprocessor*), devido à sua ampla adoção e suporte para o desenvolvimento web, oferecendo uma série de recursos e funcionalidades para processar requisi-

ções, executar operações no banco de dados e gerar respostas dinâmicas para o *front-end* (PHP.NET, 2024). Para a execução do PHP, foi utilizado o servidor XAMPP, uma solução de desenvolvimento que integra Apache e PHP, fornecendo um ambiente completo para a execução de *scripts* PHP em um servidor local (FRIENDS, 2023).

Para o armazenamento de dados, onde são persistidos os dados do sistema, foi utilizado o banco de dados *Oracle* e *PostgreSQL*.A escolha do Oracle é baseada em sua robustez, desempenho e escalabilidade, oferecendo um ambiente seguro e eficiente para armazenamento e gerenciamento de dados (ORACLE, 2024). O PostgreSQL também foi selecionado por sua flexibilidade, conformidade com padrões e forte suporte à integração de dados, proporcionando uma solução completa e confiável para as necessidades do sistema. (POSTGRESQL, 2024).

Para a administração e desenvolvimento do banco de dados Oracle, foi utilizada a ferramenta *SQL Developer*, que proporciona um ambiente integrado para criação, execução e otimização de consultas SQL. As consultas SQL foram empregadas para recuperar os dados solicitados pelo usuário a partir das requisições feitas no *front-end*, sendo posteriormente processadas pelo *back-end* e apresentadas ao usuário final em forma de tabela.

A integração dessas tecnologias proporcionou um ambiente de desenvolvimento eficiente, permitindo a construção de um sistema web funcional.

3.2 Projeto

A modelagem do sistema web desenvolvido neste trabalho foi concebida para fornecer uma interface intuitiva e eficiente para que os usuários possam interagir com o banco de dados por meio da execução de consultas SQL. A Figura 6 apresenta as etapas para o desenvolvimento de um software.

Figura 6 – O ciclo de Desenvolvimento de Software. Fonte: Adaptado de: Sigma (2024)

O acesso ao sistema é realizado por meio de um navegador web padrão, onde os usuários são direcionados para a interface principal do sistema. A partir dessa interface, os usuários têm a opção de inserir uma consulta SQL em um campo específico e enviar essa consulta para o sistema.

Conforme demonstrado na Figura 7, no *back-end*, o sistema recebe a consulta SQL enviada pelo usuário e utiliza a linguagem de programação PHP para estabelecer uma conexão com o banco de dados Oracle. Através dessa conexão, o programa PHP executa a consulta SQL no banco de dados e obtém o resultado da consulta. Esse resultado é então formatado em um formato JSON.

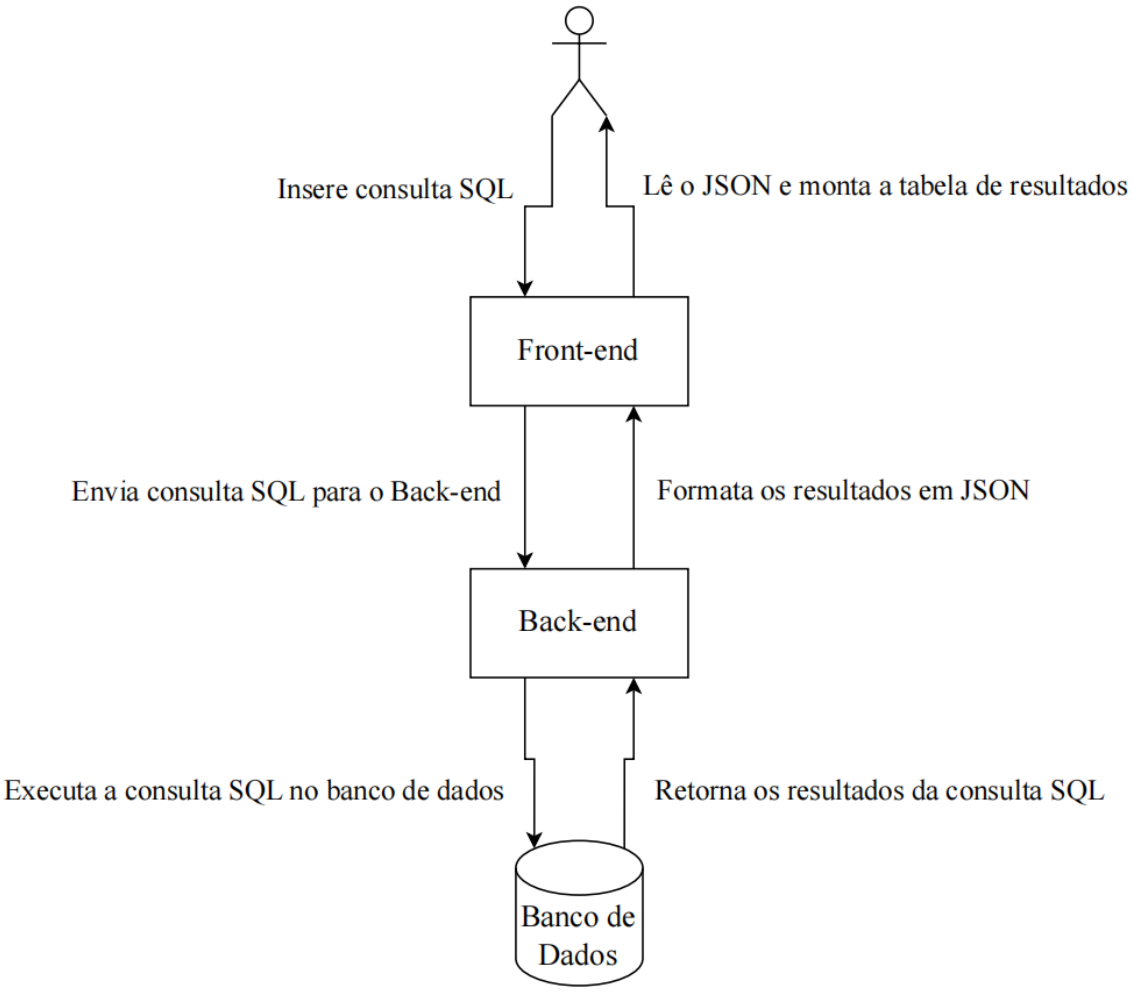


Figura 7 – Diagrama de Fluxo de Interações. Fonte: O Autor

Após a obtenção do resultado da consulta no *back-end*, o sistema retorna esse resultado em formato JSON para o *front-end*. No *front-end*, a resposta JSON é lida pelo programa JS, que é responsável por interpretar os dados e montar uma tabela dinâmica na interface do usuário. Essa tabela exibe os resultados da consulta SQL de forma organizada para o usuário final.

A arquitetura do sistema é esquematizada através do diagrama, que ilustra o pro-

cesso de interação entre o usuário, o *front-end* e o *back-end*, conforme apresentado na Figura 7. O diagrama de fluxo de dados demonstra como os dados fluem através do sistema, desde a entrada da consulta SQL pelo usuário até a exibição dos resultados na interface, descrevendo a interação entre os diferentes componentes do sistema em cada etapa do processo, desde a submissão da consulta pelo usuário até a visualização dos resultados.

Portanto, a modelagem do software adotada no desenvolvimento do sistema garante um entendimento e uma manutenibilidade do código fonte.

3.3 Implementação

Nesta seção, será descrito a implementação do sistema para resolver o problema proposto e construir a tabela de resultados.

3.3.1 PHP

O código, utilizado para realizar consultas a um banco de dados Oracle e retornar os resultados em formato JSON, utiliza a linguagem PHP em conjunto com a extensão OCI (*Oracle Call Interface*) para interagir com o banco de dados Oracle.

Assim que o usuário envia a consulta SQL, o PHP recebe a *query*, faz a devida conecção com o banco de dados e executa o consulta conforme o código a seguir:

```
1     $sql = $nome;
2     $stid = oci_parse($ss, $sql);
3     oci_execute($stid);
4
5     $data = array();
6     while ($row = oci_fetch_assoc($stid))
7         array_push($data,$row);
8
9     header('Content-Type: application/json');
10    echo json_encode($data);
```

Algoritmo 3.1 – Consultado o Banco de Dados

No Algoritmo 3.1, uma consulta SQL é atribuída à variável '$sql'. Presumivelmente, o valor de '$nome' contém a consulta a ser executada no banco de dados *Oracle*. Em seguida, o comando '*oci_parse()*' é usado para preparar a consulta SQL para execução e o comando '*oci_execute()*' é usado para executar a consulta SQL preparada.

Uma repetição *while* é usada para iterar sobre os resultados da consulta. A função '*oci_fetch_assoc()*' é usada para recuperar cada linha de resultados como uma matriz

associativa, onde as chaves são os nomes das colunas. Cada linha de resultados é adicionada à matriz '$data' utilizando a função '*array_push()*', o que resulta em um array tridimensional contendo todos os resultados da consulta.

O tipo de conteúdo da resposta é definido como JSON. Por fim, os resultados da consulta são convertidos para o formato JSON utilizando a função '*json_encode()*' e são enviados como resposta para o Java Script.

3.3.2 Java Script

As partes dos algoritmos a seguir foram desenvolvidos em Java Script onde recebem os dados retornados pela query em formato JSON. Há também, a manipulação de elementos HTML para criar uma tabela dinâmica.

```
1  // Cria a primeira linha da tabela
2      for (const key of dados){
3          if(!(coluna.includes(key[atributos[1]])) && key[atributos
               [1]] != null && key[atributos[2]] == aux){
4              const headerCell = document.createElement('th');
5              headerCell.textContent = key[atributos[1]];
6              headerRow.appendChild(headerCell);
7              coluna.push(key[atributos[1]]);
8          }
9      }
10     const headerCell2 = document.createElement('th');
11     headerCell2.textContent = 'TOTAL';
12     headerRow.appendChild(headerCell2);
13     coluna.push('TOTAL');
```

Algoritmo 3.2 – Criando a primeira linha da Tabela

O Algoritmo 3.2 está relacionado à criação da primeira linha da tabela, onde é adicionado todas as informações do primeiro atributo. Seguem algumas variáveis importantes para o entendimento do código:

- **'dados'** - Variável onde contém o JSON com as informações resgatadas do banco.
- **'atributos'** - Uma lista contendo os paramêtros solicitados na *query*, respresentando linha, coluna e profundidade.
- **'coluna'** - Uma lista onde possui todos os resultados relacionado ao segundo atributo (coluna) sem repetição.
- **'linha'** - Uma lista onde possui todos os resultados relacionado ao primeiro atributo (linha) sem repetição.

- **'aux'** - Variável onde contém a informação do terceiro atributo (profundidade) a ser plotada.

No Algoritmo 3.2 é criado o cabeçalho de uma tabela com base nas informações fornecidos em 'dados' e nas configurações de 'atributos'.

O trecho do Algoritmo 3.2 itera sobre os elementos do *array* 'dados'. Ele verifica se o valor do atributo especificado não está presente em 'coluna' (para não ser adicionados elementos repetidos), se for diferente de *null*, e se o valor do terceiro atributo (profundidade) é igual a 'aux'. Se todas essas condições forem atendidas, uma nova célula de cabeçalho é criada com o valor do atributo, adicionada à primeira linha da tabela e o valor do atributo é adicionado ao array coluna.

Por fim, é criado uma nova célula de cabeçalho com o texto 'TOTAL' e a adiciona à primeira linha da tabela e adicionado ao *array* coluna.

```
1     // Cria a primeira coluna
2     for (const key of dados) {
3         if(!(linha.includes(key[atributos[0]])) && key[atributos
              [0]] != null && key[atributos[2]] == aux){
4             row[row.length] = table.insertRow();
5             const cell = row[row.length-1].insertCell();
6             cell.textContent = key[atributos[0]];
7             linha.push(key[atributos[0]]);
8
9             for (const i in coluna) {
10                celula[celula.length] = row[row.length-1].
                      insertCell();
11                celula[celula.length-1].textContent = '-';
12            }
13        }
14    }
15    row[row.length] = table.insertRow();
16    const cell = row[row.length-1].insertCell();
17    cell.textContent = 'TOTAL';
18    for (const i in coluna) {
19        celula[celula.length] = row[row.length-1].insertCell();
20        celula[celula.length-1].textContent = '-';
21    }
```

Algoritmo 3.3 – Criando a primeira coluna da Tabela

No Algoritmo 3.3 é criado a primeira coluna da tabela com base nas informações fornecidos em 'dados' e nas configurações de 'atributos'.

O Algoritmo 3.3 itera sobre os elementos do *array* 'dados' novamente. Para cada elemento, verifica se o valor do segundo atributo especificado não está presente em 'linha', se não é *null* e se o valor do terceiro atributo (profundidade) é igual a 'aux'. Se todas essas condições forem atendidas, uma nova linha (<tr>) é adicionada à tabela e o valor do atributo é adicionado a primeira célula. As demais células da linha são preenchidas com '-'.

Após o loop, uma nova linha é adicionada à tabela para representar a linha 'TOTAL'.

```
1  // Preenche a tabela
2      for (const key of dados){
3          const indexA = linha.indexOf(key[atributos[0]]);
4          const indexB = coluna.indexOf(key[atributos[1]]);
5          if(indexA != -1 && indexB != -1 && key[atributos[2]] ==
               aux){
6              celula[(indexA*coluna.length)+indexB].textContent =
                   key[atributos[3]];
7          }
8          else if (indexA == -1 && indexB != -1 && key[atributos
               [2]] == aux){
9              celula[((linha.length)*coluna.length)+indexB].
                   textContent = key[atributos[3]];
10         }
11         else if (indexA != -1 && indexB == -1 && key[atributos
               [2]] == aux){
12             celula[(indexA*coluna.length)+coluna.length-1].
                   textContent = key[atributos[3]];
13         }
14         else if (indexA == -1 && indexB == -1 && key[atributos
               [2]] == aux){
15             celula[((linha.length)*coluna.length)+coluna.length
                   -1].textContent = key[atributos[3]];
16         }
17     }
```

Algoritmo 3.4 – Preenchendo a Tabela

Este trecho do Algoritmo 3.4 é responsável por percorrer as informações fornecidos em 'dados' e preencher as células da tabela com os valores correspondentes.

O Algoritmo 3.4 itera sobre os elementos do *array* 'dados'. Para cada elemento, o código obtém os índices da linha e da coluna correspondentes aos valores dos atributos fornecidos ('atributos[0]' e 'atributos[1]') nos arrays 'linha' e 'coluna', respectivamente.

Em seguida o código verifica várias condições para determinar em qual célula da tabela o valor deve ser inserido:

- **S**e os índices da 'linha' e da 'coluna' são válidos e o valor do terceiro atributo é igual a 'aux', o valor é inserido na célula correspondente.
- **S**e o índice da 'linha' é inválido, o índice da 'coluna' é válido e o valor do terceiro atributo é igual a 'aux', o valor é inserido na célula correspondente à linha 'TOTAL'.
- **S**e o índice da 'linha' é válido, o índice da 'coluna' é inválido e o valor do terceiro atributo é igual a 'aux', o valor é inserido na célula correspondente à última coluna da linha, 'TOTAL'.
- **S**e tanto o índice da 'linha' quanto o índice da 'coluna' são inválidos e o valor do terceiro atributo é igual a 'aux', o valor é inserido na célula correspondente à interseção da última linha com a última coluna.

Os códigos apresentados representam as principais partes do código responsáveis pelo funcionamento e criação da tabela. Dessa forma, ao abordar cada código e função, é possível a compreensão do processo de implementação e o funcionamento do sistema.

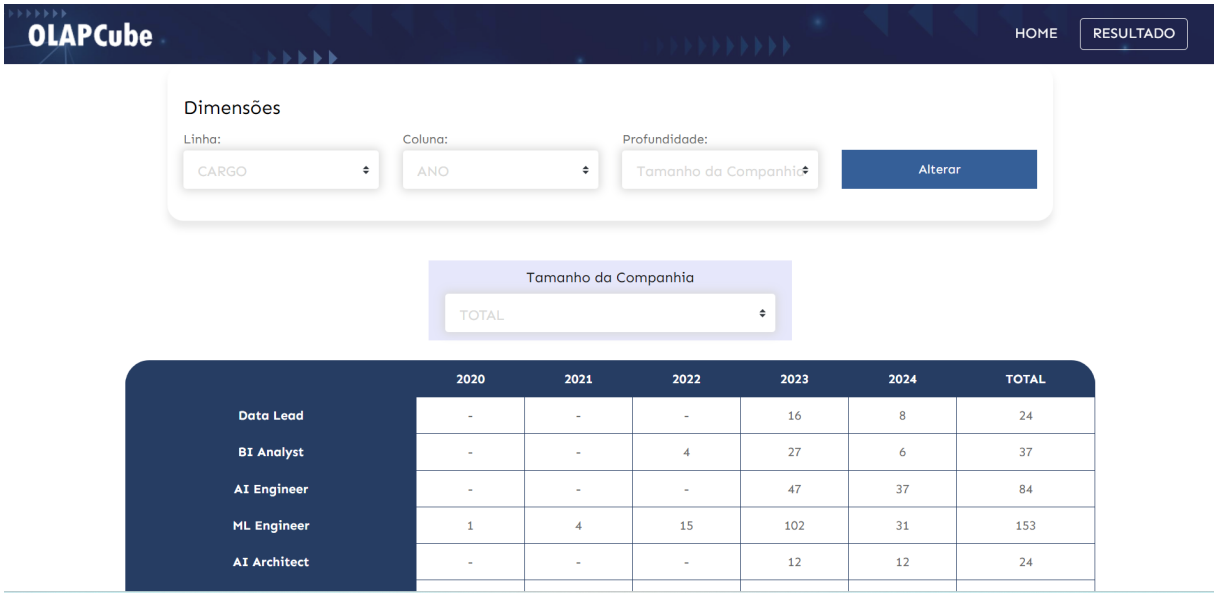
3.4 Interface e Funcionamento

O sistema desenvolvido é composto por duas páginas distintas, cada uma com funcionalidades específicas para facilitar a consulta e a análise dos resultados. A primeira página, denominada página inicial, serve como ponto de partida para os usuários realizarem suas consultas.

Figura 8 – Página Inicial. Fonte: O Autor

Conforme a Figura 8, na página inicial, os usuários são recebidos com uma mensagem instrutiva, orientando-os sobre o uso do sistema e incentivando-os a inserir sua consulta. Logo abaixo da mensagem, encontra-se um campo de entrada de texto onde os usuários podem digitar a consulta desejada. Ao lado deste campo, há um botão designado para enviar a consulta inserida.

A segunda página é dedicada à apresentação dos resultados da consulta realizada pelos usuários. Conforme a Figura 9, ela oferece uma interface para análise e manipulação dos dados retornados.



	2020	2021	2022	2023	2024	TOTAL
Data Lead	-	-	-	16	8	24
BI Analyst	-	-	4	27	6	37
AI Engineer	-	-	-	47	37	84
ML Engineer	1	4	15	102	31	153
AI Architect	-	-	-	12	12	24

Figura 9 – Apresentação da Tabela. fonte: O Autor

No topo da página, os usuários têm acesso a uma seção que permite a análise das dimensões dos resultados da consulta. Estas dimensões são apresentadas em um campo *select*, onde é possível escolher a dimensão que deseja analisar (linha, coluna e profundidade), conforme a Figura 10. Ao lado deste campo *select*, encontra-se um botão que ao ser clicado exibe a tabela com as informações atualmente selecionadas para a respectiva dimensão.

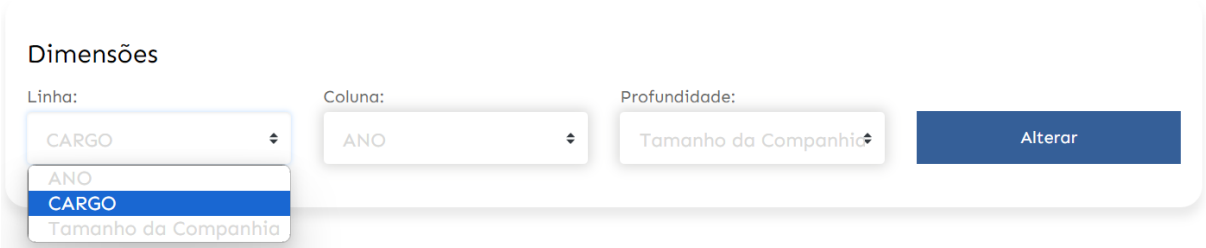


Figura 10 – Campos de Dimensões: Fonte: O Autor

Logo abaixo da seção de análise de dimensões, os resultados da consulta são exibidos em forma de tabela bidimensional. Esta tabela leva em consideração os campos

escolhidos para as dimensões de linha e coluna. Os dados apresentados são organizados de acordo com a estrutura da consulta realizada pelo usuário.

Acima da tabela de resultados , os usuários têm acesso a outro campo *select* (Figura 11) que permite a seleção do parâmetro da terceira dimensão (profundidade) relacionado à tabela apresentada. Este campo oferece a flexibilidade de alterar automaticamente a informação apresentada na tabela, de acordo com a escolha feita pelo usuário.

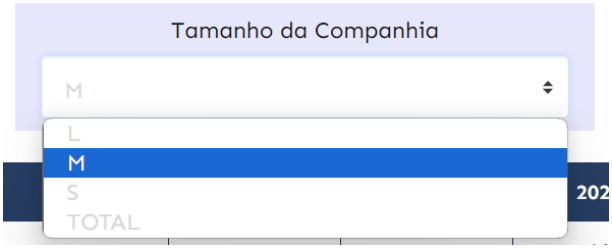


Figura 11 – Campo de Profundidade

Além da tabela de resultados apresentada na Figura 9, logo abaixo, encontra-se outra tabela que representa diretamente os dados retornados pelo banco de dados, como ilustrado na Figura 12. Essa tabela fornece uma visão mais detalhada dos resultados, permitindo que os usuários compreendam melhor a estrutura dos dados e a correspondência com a tabela apresentada acima. Ao visualizar diretamente os dados brutos retornados pelo banco, os usuários podem esclarecer dúvidas, verificar a precisão dos resultados e facilitar a compreensão da transformação desses dados na tabela de resultados apresentada anteriormente.

TABELA RETORNADA PELO BANCO DE DADOS

LOJA	ANO	FERIADO	ROUND(SUM(WEEKLY_SALES),3)
NULL	NULL	NULL	6737217020
NULL	NULL	TRUE	505299584
NULL	NULL	FALSE	6231919100
NULL	2010	NULL	2288885760
NULL	2010	TRUE	200224608
NULL	2010	FALSE	2088660860
NULL	2011	NULL	2448200960
NULL	2011	TRUE	206735488
NULL	2011	FALSE	2241465340
NULL	2012	NULL	2000132350
NULL	2012	TRUE	98339480
NULL	2012	FALSE	1901792900

Figura 12 – Tabela retornada pelo Banco de Dados

Essa estrutura de interface e funcionamento proporciona aos usuários uma experiência eficiente na realização de consultas e análise de dados, permitindo uma interação

dinâmica com o sistema desenvolvido.

3.5 Base de Dados

Nesta seção, é descrita a base de dados utilizada para a realização deste trabalho, incluindo a fonte do conjunto de dados, o processo de importação para o banco de dados Oracle e outras considerações relevantes.

O conjunto de dados utilizado neste sistema é o "*Retail Dataset*", disponível no repositório do Kaggle (2024). Este conjunto de dados contém informações de quarenta e cinco lojas sobre vendas no varejo, incluindo dados de vendas, informações sobre lojas e características econômicas. O dataset é composto por três arquivos CSV (*Comma-separated values*), cada um contendo diferentes aspectos dos dados, como informações sobre vendas, informações sobre as lojas, entre outros.

Para utilizar os dados contidos no arquivo CSV dentro do ambiente do banco de dados Oracle, foi necessário realizar um processo de importação dos dados. O Oracle SQL Developer facilita a importação dados em uma tabela existente. Para isso foram criadas as tabelas com seus devidos atributos.

A primeira tabela é a '*stores*', que contém dados relacionados as lojas, o tipo e seu tamanho. A tabela possui os seguintes campos:

- *Store* - representa o número da loja, sendo a chave primária da tabela.
- *Type* - representa o tipo da loja, contendo os valores A, B e C.
- *Size* - contém um valor númerico representando o tamanho da loja.

O Algoritmo 3.5 apresenta o código utilizado para criação desta tabela.

```
1 CREATE TABLE stores (
2     Store INT PRIMARY KEY,
3     Type CHAR,
4     Size VARCHAR2(20)
5 );
```

Algoritmo 3.5 – Criação Tabela 'stores'

A segunda tabela é a '*features*', que possui informações de características das lojas e da região em que as mesmas se encontram. Dentre as características estão: uma data da semana, temperatura média na região, preço do combustível na região, taxa de desemprego, se a semana é uma semana especial de férias, CPI (Índice de Preços ao Consumidor) etc.

- *Store* - representa o número da loja, sendo uma chave estrangeira da tabela '*stores*'.
- *Date* - Uma data representando uma semana.
- *Fuel_Price* - custo do combustível na região.
- *MarkDown1-5* - dados anônimos relacionados a promoções. Os dados do *MarkDown* não estão disponíveis para todas as lojas o tempo todo. Qualquer valor em falta é marcado com um NA.
- CPI - índice de preços ao consumidor.
- *Unemployment* - taxa de desemprego
- *IsHoliday* - se a semana é uma semana especial de férias ou feriado, contendo os valores *TRUE* ou *FALSE*.

O Algoritmo 3.6 apresenta o código utilizado para criação desta tabela:

```
1  CREATE TABLE features (
2      Store INT,
3      Date DATE,
4      Temperature BINARY_FLOAT,
5      Fuel_Price BINARY_FLOAT,
6      MarkDown1 VARCHAR2(20),
7      MarkDown2 VARCHAR2(20),
8      MarkDown3 VARCHAR2(20),
9      MarkDown4 VARCHAR2(20),
10     MarkDown5 VARCHAR2(20),
11     CPI BINARY_DOUBLE,
12     Unemployment VARCHAR2(20),
13     IsHoliday VARCHAR2(10),
14     PRIMARY KEY (Store,Date),
15     CONSTRAINT fk_store_features FOREIGN KEY (Store) REFERENCES
          stores (Store)
16 );
```

Algoritmo 3.6 – Criação Tabela 'features'

E por fim, a terceira tabela é a '*sales*' onde possui dados históricos de vendas, que abrangem os anos de 2010 a 2012.

- *Store* - representa o número da loja, sendo uma chave estrangeira da tabela '*stores*'.
- *Dept* - representa o número do departamento

- *Weekly_Sales* - contém um valor númerico representando as vendas semanais da loja.
- *IsHoliday* - se a semana é uma semana especial de férias ou feriado, contendo os valores *TRUE* ou *FALSE*.

O código utilizado para a criação ta tabela é apresentado no Algoritmo 3.7.

```
1 CREATE TABLE sales (
2     Store INT,
3     Dept INT,
4     Date DATE,
5     Weekly_Sales BINARY_FLOAT,
6     IsHoliday VARCHAR2(10),
7     PRIMARY KEY (Store,Dept,Date),
8     CONSTRAINT fk_store_sales FOREIGN KEY (Store) REFERENCES
          stores (Store)
9 );
```

Algoritmo 3.7 – Criação Tabela 'sales'

Após a criação das tabelas, conforme Figura 13, é possível realizar a importação dos dados.

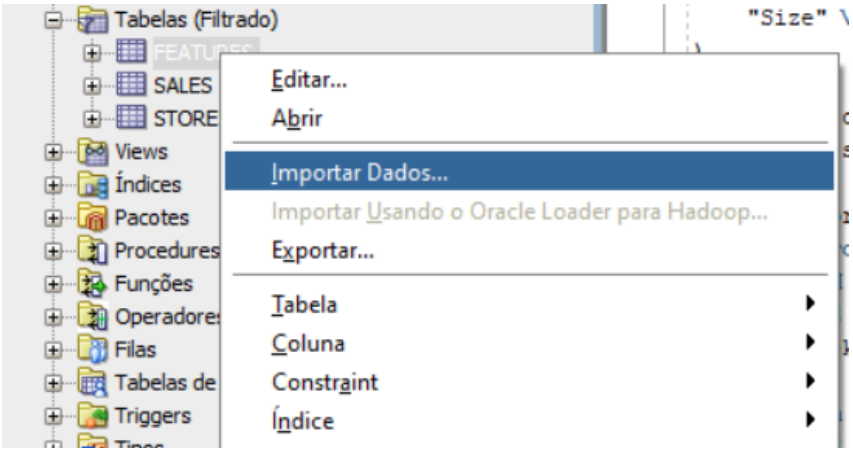


Figura 13 – Importação de dados no Oracle SQL Developer. Fonte: O Autor

O Oracle *SQL Developer* oferece uma funcionalidade de importação de dados que pode ser utilizada para importar arquivos CSV diretamente para o banco de dados *Oracle*. Isso pode ser feito selecionando a opção de importação de dados no menu do *SQL Developer* e seguindo as instruções para selecionar o arquivo CSV e mapear as colunas para as tabelas correspondentes no banco de dados.

Por meio deste processo de importação, os dados contidos no arquivo CSV são integrados ao ambiente do banco de dados *Oracle*, proporcionando uma base para a realização das análises e estudos propostos neste trabalho.

3.6 Experimentos

Nesta seção, serão apresentados os experimentos realizados no sistema desenvolvido ao longo deste trabalho, bem como os resultados obtidos durante esses testes. Os testes têm como objetivo avaliar o desempenho e a funcionalidade do sistema em diferentes cenários e condições de uso.

3.6.1 Experimentos 1

Descreveremos o primeiro teste realizado no sistema desenvolvido. O teste envolveu a execução de uma consulta SQL específica no sistema, seguida pela análise dos resultados obtidos. Para o primeiro teste, foi desenvolvida a seguinte consulta SQL:

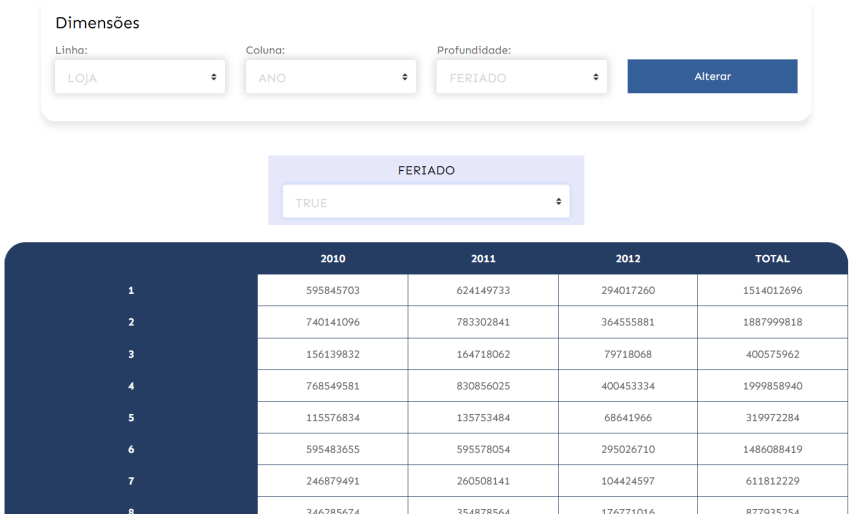
SELECT Store as Loja, Ano, IsHoliday as Feriado,

SUM(Weekly_Sales)

FROM sales

GROUP BY CUBE (Store, to_char(Date, 'yyyy') as Ano, IsHoliday)

Esta consulta realiza uma análise detalhada das vendas na tabela '*sales*', filtrando por loja, ano e feriado, e agrupando os resultados pela soma das vendas semanais. Os principais objetivos deste teste incluem identificar qual loja teve o melhor desempenho em termos de vendas, em qual ano as vendas foram mais altas e se as vendas durante os feriados foram significativamente mais altas em determinado ano.



	2010	2011	2012	TOTAL
1	595845703	624149733	294017260	1514012696
2	740141096	783302841	364555881	1887999818
3	156139832	164718062	79718068	400575962
4	768549581	830856025	400453334	1999858940
5	115576834	135753484	68641966	319972284
6	595483655	595578054	295026710	1486088419
7	246879491	260508141	104424597	611812229
8	346285674	354878564	176771016	877935254

Figura 14 – Primeiro Teste

A consulta SQL foi enviada ao sistema através do site desenvolvido, que processou a consulta e retornou a tabela resultante, conforme mostrado na Figura 14.

Os resultados obtidos foram analisados. A loja que apresentou o melhor desempenho de vendas foi a Loja 20 com um número de 301.397.600. O ano mais lucrativo foi 2011 com soma das vendas semanais totalizando 2.448.200.190, conforme a Tabela 2. Além disso, foi possível analisar separadamente o rendimento de cada loja em relação aos feriados, conforme a Tabela 3.

	2010	2011	2012	TOTAL
		...		
20	101.733.048	109.836.848	89.827.728	301.397.600
		...		
33	12.766.822	12.957.837	11.435.557	37.160.216
TOTAL	2.288.886.270	2.448.200.190	2.000.132.740	6.737.217.540

Tabela 2 – Tabela Teste 1 (Loja x Ano x Feriado) - Fonte: O Autor

Na Tabela 2, podemos vizualizar as lojas com os resultados do maior e do menor número de vendas total, além do total geral de cada ano. Lembrando que a terceira dimensão está como o 'Feriado' com a opção 'TOTAL' selecionado, ou seja, a tabela mostra o resultado de cada loja em cada ano tanto para datas de feriado (*TRUE*) como para datas que não (*FALSE*).

Na Tabela 3, vizualizamos as vendas de cada loja em relação as semanas de feriado, tendo na terceira dimensão 'Feriado' com a opção '*TRUE*' selecionada. Os anos de 2010 e 2011 tiveram um melhor desempenho de vendas nas semanas de feriado comparado ao ano 2012 que teve um total de vendas consideravelmente menor com 98.339.480.

	2010	2011	2012	TOTAL
1	6.472.363	6.720.870,5	3.464.245	16.657.479
2	8.386.098,5	8.404.470	4.002.099,25	20.792.668
		...		
TOTAL	200.224.608	206.735.488	98.339.480	505.299.576

Tabela 3 – Tabela Teste 1 (Loja x Ano x Feriado = *TRUE*)- Fonte: O Autor

Este teste inicial forneceu uma base para a avaliação do sistema desenvolvido e serviu como ponto de partida para testes subsequentes.

3.6.2 Experimentos 2

O segundo teste realizado no sistema desenvolvido durante este trabalho envolveu a criação da seguinte consulta SQL:

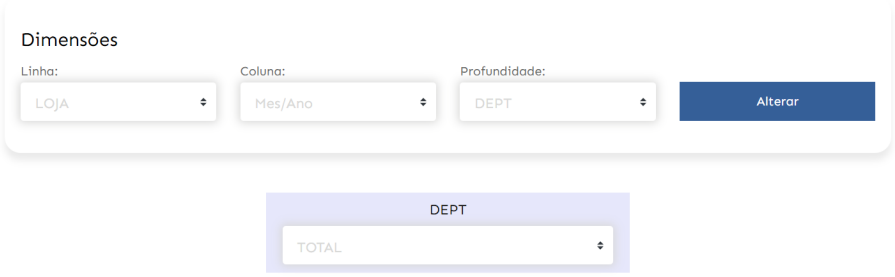
SELECT Store as Loja, "Mes/Ano", Dept,

ROUND(AVG(Weekly_Sales), 3)

FROM Sales

GROUP BY CUBE(Store, to_char(Date,'mm/yyyy'), dept)

Esta consulta realiza uma análise detalhada das médias de vendas semanais por loja, mês/ano e departamento na tabela "sales". Os principais objetivos deste teste incluem identificar qual mês, loja e departamento teve a melhor média de vendas. Além disso, a consulta permite verificar quanto cada loja e departamento vendeu em determinado mês.



	01/2011	01/2012	02/2010	02/2011	02/2012	03/2010	03/2011	03/2012	04/2010	04/2011	04/2012	05/2010	05/2011	05/2012	06/2010	06/2011
1	19161,016	20012,906	21749,463	22299,262	23441,641	20245,842	21749,572	22974,779	20561,779	21659,502	22846,369	20880,063	21353,42	22780,785	21728,863	21813,279
2	24044,986	24100,766	28399,816	27818,695	27921,082	26384,074	25963,565	26725,238	26667,44	25591,256	26561,875	27289,072	24955,648	27307,078	28615,752	26600,023
3	5876,675	5784,975	6658,358	6769,067	7003,008	6018,355	6407,267	6733,813	5868,059	6038,935	6876,315	5886,59	6012,976	6760,765	5893,637	6351,171
4	25974,174	27511,32	28132,34	30318,902	31952,559	25771,75	28387,246	30096,699	25464,756	28029,486	30499,695	26798,277	28014,607	30370,566	26774,107	29083,801
5	4336,165	4673,352	4679,876	5068,574	5199,37	4564,556	4782,264	5324,492	4996,215	4922,235	5572,017	5027,631	4728,901	5491,949	5075,594	5007,506

Figura 15 – Segundo Teste. Fonte: O Autor

A consulta SQL foi enviada ao sistema através do site desenvolvido, que processou a consulta e retornou a tabela resultante contendo os dados analisados, conforme mostrado na Figura 15.

Os resultados obtidos foram analisados. Foram identificadas tendências e padrões nos dados, conforme mostrado na Tabela 4, incluindo qual loja teve a média de vendas mais alta por mês sendo a loja 14 no mês de Dezembro de 2010 com média de 37.433,441, qual mês com média de vendas mais alta sendo também Dezembro de 2010 com média de 19.570,365 e, por fim, qual a loja com média de vendas mais baixa sendo a loja 5 no mês de Janeiro de 2011 com média de 4.336,165.

Este teste forneceu informações adicionais para complementar a análise do sistema desenvolvido e garantir sua eficácia.

	12/2010	01/2011	TOTAL
...			
5	5.734,463	4.336,165	5.053,413
...			
14	37.433,441	25.166,359	28.784,854
TOTAL	19.570,365	13.997,781	15.980,782

Tabela 4 – Tabela Teste 2 (Loja x Mes/Ano x Departamento)- Fonte: O Autor

3.6.3 Experimentos 3

Nesta subseção, descreveremos o terceiro teste realizado no sistema desenvolvido durante este trabalho. O experimento envolveu a execução de uma consulta SQL específica no sistema, seguida pela análise dos resultados obtidos.

SELECT Tamanho, Type as Tipo, Store as Loja, COUNT(*)
FROM Stores GROUP BY CUBE
(CASE WHEN Size < 100000 THEN 'TAM < 100000' ELSE
(CASE WHEN Size>100000 AND Size<200000 then '100000>TAM
<200000' ELSE 'TAM > 200000' END) END) as Tamanho,
Type, Store)

Esta consulta realiza uma análise detalhada dos tamanhos das lojas em relação aos tipos, agrupando os resultados pela quantidade total de lojas de cada tamanho e tipo. Os principais objetivos deste teste incluem identificar qual tipo de loja prevalece em relação ao tamanho das lojas.

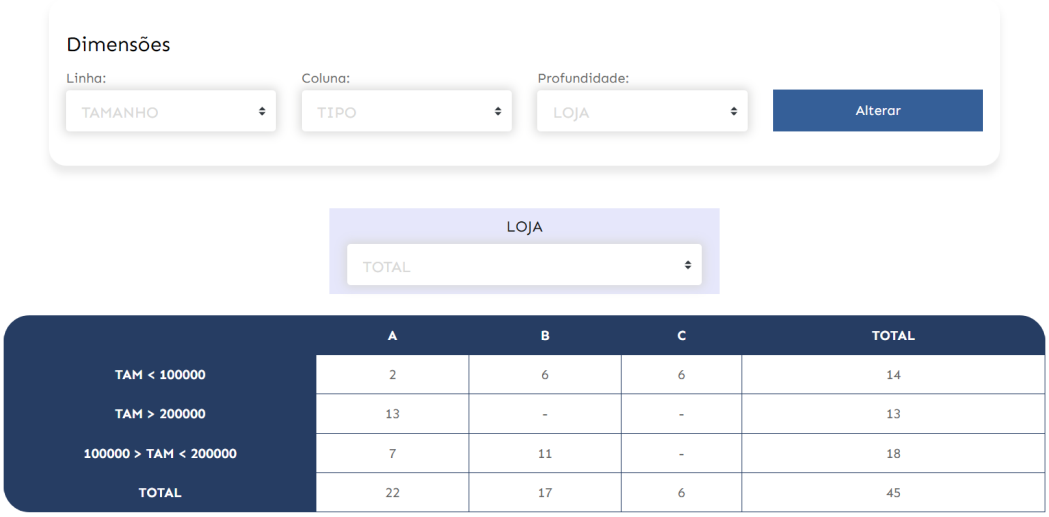


Figura 16 – Terceiro Teste. Fonte: O Autor

A consulta SQL foi enviada ao sistema através do site desenvolvido, que processou a consulta e retornou a tabela resultante contendo os dados, conforme a Figura 16.

De acordo com os resultados, a loja com o tipo 'A' prevalece com 22 lojas no total. As lojas com tamanho entre 100000 e 200000 estão em maior número com 18 lojas. É possível indentificar também que as lojas do tipo C estão em menor número, e todas tem seu tamanho menor que 100000.

Com base nos resultados dos testes, foi possível avaliar o desempenho do sistema em atender aos requisitos estabelecidos. Quaisquer informações derivadas dos testes podem ajudar a estabelecer possíveis ajustes e melhorias.

4 Conclusão

Ao finalizar este trabalho, é possível concluir que o objetivo principal de desenvolver uma ferramenta web capaz de visualizar resultados tridimensionais a partir de consultas SQL com agregação *GROUP BY CUBE* foi alcançado.

O projeto foi concebido e executado conforme o planejado, abordando desde o preparo do ambiente e das ferramentas necessárias até a realização de testes em uma base de dados real.

Durante o desenvolvimento, houveram alguns desafios, especialmente na configuração do ambiente com o banco de dados e nas ferramentas utilizadas. Esses aspectos exigiram um esforço adicional e um maior tempo de dedicação, mas foram superados com sucesso, contribuindo para o aprendizado e crescimento profissional ao longo do projeto.

Uma vez implementado, o sistema demonstrou sua utilidade e eficácia na análise de dados tridimensionais, fornecendo resultados que podem auxiliar na tomada de decisões estratégicas, não só em organizações, mas também na área acadêmica auxiliando professores e alunos a compreender, através da vizualização, as consultas de agregação usando o *GROUP BY*.

Dessa forma, o sistema desenvolvido foi disponibilizado como um projeto de código aberto no Vieira (2024) para promover o software livre e incentivar a colaboração da comunidade. Ao adotar essa abordagem, busca-se não apenas proporcionar acesso livre ao código-fonte, mas também fomentar um ambiente de aprendizado colaborativo.

Em suma, este trabalho não apenas alcançou seus objetivos, mas também proporcionou uma experiência enriquecedora de aprendizado e desenvolvimento. Através dele, foi possível compreender melhor a importância e o potencial das ferramentas de OLAP na análise de dados organizacionais.

Que o trabalho possa servir como um ponto de partida para futuras pesquisas e projetos na área de visualização e análise de dados.

4.1 Trabalhos Futuros

Apesar da conclusão do projeto, pode-se reconhecer que ainda há espaço para aprimoramentos futuros. O sistema está em estágio inicial, então considera-se a adição de novas funcionalidades para enriquecimento do trabalho.

Dentre as possíveis melhorias, destaca-se :

- A expansão do sistema para aceitar outros tipos de consultas SQL, como o GROUP BY ROLLUP.
- A melhoria do design para uma representação mais eficiente e intuitiva do cubo de dados, possibilitando a visualização da terceira dimensão (profundidade) de maneira mais ampla.
- Oferecer uma interface gráfica, para aqueles sem conhecimento prévio em SQL, para gerar consultas de maneira intuitiva, onde o usuário poderia arrastar e selecionar atributos, especificar condições e definir agrupamentos, resultando na construção automática da consulta SQL.
- Capacidade de aumentar ou diminuir o número de dimensões conforme necessário e solicitado na *query*.
- Adição de tutoriais explicativos que ensinem sobre o funcionamento do *GROUP BY CUBE* e sobre a utilização do sistema desenvolvido. Esses tutoriais seriam uma ferramenta valiosa para os usuários que desejam aprender mais sobre o processo de análise de dados tridimensionais e como utilizar o sistema de forma eficaz em seus contextos específicos..

Referências

BOURHIS, P.; REUTTER, J. L.; VRGOč, D. Json: Data model and query languages. **Information Systems**, v. 89, p. 101478, 2020. ISSN 0306-4379. doi:10.1016/j.is.2019.101478. Citado na página 20.

BRAY, T. **The JavaScript Object Notation (JSON) Data Interchange Format**. RFC Editor, 2014. RFC 7159. (Request for Comments, 7159). Disponível em: <https://www.rfc-editor.org/info/rfc7159>. Citado 2 vezes nas páginas 20 e 21.

ELMASRI, R.; NAVATHE, S. B. Sistemas de banco de dados. Pearson Addison Wesley São Paulo, 2014. Citado 4 vezes nas páginas 9, 12, 17 e 18.

FRIENDS, A. Xampp. 2023. Disponível em: <https://www.apachefriends.org/index.html>. Citado na página 23.

GRAY, J.; CHAUDHURI, S.; BOSWORTH, A.; LAYMAN, A.; REICHART, D.; VENKATRAO, M.; PELLOW, F.; PIRAHESH, H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. **Data mining and knowledge discovery**, Springer, 1997. doi:10.1023/A:1009726021843. Citado 5 vezes nas páginas 5, 13, 14, 15 e 16.

ISO. **ISO/IEC 9075:1992: Title: Information technology — Database languages — SQL**. International Organization for Standardization, 1992. 587 p. Disponível em: <https://www.iso.org/standard/16663.html>. Citado na página 13.

KAGGLE. **Retail Data Analytics**. 2024. Disponível em: <https://www.kaggle.com/datasets/manjeetsingh/retaildataset?resource=download&select=stores+data-set.csv>. Citado na página 32.

MCPEAK, J. **Beginning JavaScript**. Fifth edition. Newark: Wiley, 2015. (Wrox programmer to programmer). ISBN 1118903749. Citado na página 20.

Mozilla contributors. Css. **Mozilla Corporation's**, 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Citado na página 22.

_____. Html: Linguagem de marcação de hipertexto. **Mozilla Corporation's**, 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Citado na página 22.

ORACLE. **Oracle Database**. 2024. Disponível em: <https://www.oracle.com/database/>. Citado na página 23.

PHP.NET. Php: Hypertext preprocessor. 2024. Disponível em: <https://www.php.net/>. Citado na página 23.

POSTGRESQL. **PostgreSQL: The World's Most Advanced Open Source Relational Database**. 2024. Disponível em: <https://www.postgresql.org/>. Citado na página 23.

RAINARDI, V. **Building a data warehouse: with examples in SQL Server**. Berkeley, CA: Apress, 2008. doi:10.1007/978-1-4302-0528-9. ISSN 978-1-4302-0528-9. Citado na página 17.

SHERMAN, R. **Business intelligence guidebook: From data integration to analytics**. Boston: Morgan Kaufmann, 2014. doi:10.1016/B978-0-12-411461-6.00016-2. ISBN 978-0-12-411461-6. Citado na página 9.

SIGMA. **Software Development**. 2024. Disponível em: <https://www.sigma-emea.com/services/software-development/>. Citado na página 23.

SUMATHI, S.; SIVANANDAM, S. Data warehousing, data mining, and olap. **Introduction to Data Mining and its Applications**, Springer, 2006. doi:10.1007/978-3-540-34351-6_2. Citado na página 10.

TANIAR, D.; RAHAYU, W. **Data Warehousing and Analytics: Fueling the Data Engine**. Data-Centric Systems and Applications: Springer Nature, 2022. doi:10.1007/978-3-030-81979-8. Citado 5 vezes nas páginas 9, 10, 16, 17 e 18.

VAISMAN, A.; ZIMÁNYI, E. Data warehouse systems. **Data-Centric Systems and Applications**, Springer, 2014. doi:10.1007/978-3-642-54655-6. Citado 7 vezes nas páginas 9, 12, 13, 16, 17, 18 e 19.

VIEIRA, G. F. **OLAPCube**. 2024. Disponível em: <https://github.com/GabrielFernandesVieira/OLAPCube>. Citado na página 40.

W3S. Bootstrap get started. **W3Schools**, 2021. Disponível em: <https://www.w3schools.com/bootstrap/bootstrap_get_started.asp>. Citado na página 22.

Apêndices

APÊNDICE A – Arquivo *JavaScript* - script.js

No Apêndice A é apresentado o código JavaScript utilizado no desenvolvimento do sistema, denominado 'script.js'. Este arquivo contém as principais funções responsáveis pela interação entre o *front-end* e o *back-end*, proporcionando uma experiência fluida aos usuários.

A primeira função, apresentada no Algoritmo A.1, denominada *'fetchData'*, é responsável por realizar uma requisição HTTP para obter os dados resultantes de uma consulta SQL feita no *back-end*. Ela recebe como parâmetro o arquivo que contém a lógica para realizar a consulta SQL. Após obter a resposta da requisição, verifica se a resposta foi bem-sucedida. Se sim, converte os dados recebidos para o formato JSON e então chama outras funções para criar uma tabela com os dados retornados e disponibilizá-la na interface do usuário. Em caso de erro durante a requisição, um erro é lançado e exibido no console.

```
function fetchData(arq) {
    fetch(arq)
        .then(response => {
            if (!response.ok) {
                throw new Error('Erro na solicitacao HTTP: ' +
                    response.status);
            } return response.json(); })
        .then(data => {
            window.parametros = [];
            for (const key in data[0]) parametros.push(key);
            window.tabela = data;
            criarCampoSelect(parametros);
            criaCampoDimensao(tabela,2);
            buildTable(tabela,parametros,null);
            tabelaBD(tabela);
        })
        .catch(error => {
            console.error('Erro durante a solicitacao fetch:',
                error); });
}
```

Algoritmo A.1 – Carregando dados da query SQL

Conforme apresentado no Algoritmo A.2, a função 'tabelaBD' é responsável por criar uma tabela HTML e preenchê-la com os dados obtidos do banco de dados. Primeiro, ela obtém o contêiner onde a tabela será exibida. Em seguida, cria o cabeçalho da tabela, inserindo os nomes das colunas. Posteriormente, preenche a tabela com os dados retornados, inserindo cada item em uma linha e cada propriedade do item em uma célula. Por fim, adiciona a tabela ao contêiner especificado.

```
function tabelaBD(data) {
    const tableContainer = document.getElementById('tablebd-
       container');
    const table = document.createElement('table');
    // Cria o cabecalho da tabela
    const headerRow = table.insertRow(0);
    for (const key in data[0]) {
        const headerCell = document.createElement('th');
        headerCell.textContent = key;
        headerRow.appendChild(headerCell);
    }
    // Preenche os dados na tabela
    for (const item of data) {
        const row = table.insertRow();
        for (const key in item) {
            const cell = row.insertCell();
            if(item[key] == null) cell.textContent = 'NULL';
            else cell.textContent = item[key];
        }
    }
    // Adiciona a tabela ao conteiner
    tableContainer.innerHTML = '';
    tableContainer.appendChild(table);
}
```

Algoritmo A.2 – Criando tabela do Banco de Dados

A função criarCampoSelect, no Algoritmo A.3, é responsável por criar e preencher os campos de seleção na interface do usuário. Ela recebe uma lista de opções (no caso, os nomes dos parâmetros da tabela) e preenche os campos de seleção correspondentes. No final, define os valores padrão para os campos de seleção.

```
function criarCampoSelect(languagesList) {
    const languagesSelectx = document.getElementById("x");
    const languagesSelectz = document.getElementById("z");
    const languagesSelecty = document.getElementById("y");

```

```
6       for (var language = 0; language < languagesList.length-1;
            language++){
7           option = new Option(languagesList[language], language);
8           languagesSelecty.options[language] = option;
9       }
10      for (var language = 0; language < languagesList.length-1;
            language++){
11          option = new Option(languagesList[language], language);
12          languagesSelectx.options[language] = option;
13      }
14      for (var language = 0; language < languagesList.length-1;
            language++){
15          option = new Option(languagesList[language], language);
16          languagesSelectz.options[language] = option;
17      }
18      languagesSelectx.value = 1;
19      languagesSelecty.value = 0;
20      languagesSelectz.value = 2;
21  }
```

Algoritmo A.3 – Criando campos Linha x Coluna x Profundidade

Já no Algoritmo A.4, a função 'criaCampoDimensao' é responsável por criar e preencher um campo de seleção específico na interface do usuário, com base nos dados retornados pelo banco de dados. Ela recebe os dados e um atributo que indica qual dimensão será analisada. A função então cria opções para o campo de seleção com base nos valores únicos encontrados na dimensão escolhida, além de uma opção adicional para exibir o total. Também atualiza o elemento de texto na interface para indicar qual dimensão está sendo analisada.

```
1   function criaCampoDimensao(dados,atributo){
2       const languagesSelecta = document.getElementById("a");
3       const languageH = document.getElementById("texto");
4       var coluna = [];
5       var language = 0;
6       // Removendo todas as opcoes
7       while (languagesSelecta.firstChild) {
8           languagesSelecta.removeChild(languagesSelecta.firstChild)
9       }
10      for (const key of dados){
11          if(!(coluna.includes(key[parametros[atributo]])) && key[
                parametros[atributo]] != null){
12              coluna.push(key[parametros[atributo]]);
```

```
13              option = new Option(key[parametros[atributo]], key[
                    parametros[atributo]]);
14              languagesSelecta.options[language] = option;
15              language++;
16          }
17      }
18      option = new Option("TOTAL", "TOTAL");
19      languagesSelecta.options[language] = option;
20      languagesSelecta.value = "TOTAL";
21      languageH.textContent = parametros[atributo];
22  }
```

Algoritmo A.4 – Criando o campo para seleção do parâmetro da profundidade

Na a função 'alterarTabela', apresentada no Algoritmo A.5, é responsável por atualizar dinamicamente a tabela exibida na interface do usuário com base nas seleções feitas nos campos de seleção. Ela obtém os valores selecionados nos campos de seleção correspondentes às dimensões da tabela e, em seguida, reconstrói a tabela com base nesses valores.

```
1  function alterarTabela(){
2      var field_x = document.getElementById("x");
3      var field_y = document.getElementById("y");
4      var field_z = document.getElementById("z");
5      criaCampoDimensao(tabela, field_z.value);
6      buildTable(tabela, [parametros[field_y.value]].concat([
           parametros[field_x.value]].concat([parametros[field_z.value
           ]].concat(parametros[3]))), null);
7  }
```

Algoritmo A.5 – Alterando dimensões da tabela

O Algoritmo A.6 é responsável por inicializar o processo de obtenção e exibição dos dados na interface do usuário. Ela obtém os dados salvos localmente (consulta SQL), realiza uma requisição para obter os dados do banco de dados e inicia o processo de construção da interface com base nesses dados.

```
1  var registro = localStorage.getItem("storage");
2  var parse = JSON.parse(registro);
3  var obj = JSON.parse(parse);
4
5  fetchData('js/fetchdata.php?txtnome=' + obj.name);
```

Algoritmo A.6 – Main

APÊNDICE B – Arquivo *JavaScript* - localStorage.js

No Apêndice B é apresentado o código JavaScript utilizado no desenvolvimento do sistema, denominado *'localStorage.js'*. Este arquivo contém uma função autoinvocável que permite armazenar temporariamente dados localmente no navegador do usuário.

```
(function () {
    function adicionar() {
        var dados = JSON.stringify({
            name    : document.querySelector("textarea[name=
               txtnome]").value
        });
        // tbClientes.push(dados);
        localStorage.setItem("storage", JSON.stringify(dados));
        return true;
    }
    var form = document.querySelector("form");
    form.addEventListener("submit", function () {
        // event.preventDefault(); event
        return adicionar();
    });
})();
```

Algoritmo B.1 – localStorage

Esta função, apresentada no Algoritmo B.1 é responsável por obter os dados inseridos pelo usuário em um campo de texto (consulta SQL) e armazená-los temporariamente no armazenamento local do navegador. Primeiro, ela utiliza o método querySelector para selecionar o campo de texto e extrair seu valor. Em seguida, os dados são estruturados em um objeto JavaScript com o valor do campo de texto. Esses dados são então convertidos em uma string JSON por meio do método JSON.stringify para serem armazenados no *localStorage*. O método setItem é então usado para armazenar esses dados com a chave "storage"no localStorage.

O código adiciona um ouvinte de evento ao formulário, que será acionado quando o formulário for enviado. Quando o formulário é enviado, a função adicionar é chamada para armazenar os dados no *localStorage*.

APÊNDICE C – Arquivo *PHP* - fetchdata.php

No Apêndice C é apresentado o código PHP utilizado no desenvolvimento do sistema, denominado '*fetchdata.php*'. Este arquivo contém a lógica responsável por realizar consultas SQL no banco de dados e retornar os resultados no formato JSON para o *front-end*.

```php
<?php
if (isset($_GET["txtnome"])){
    $sql = $_GET["txtnome"];

    $servidor     = "localhost:****/******";
    $usuario1     = "*******";
    $senha1       = "*******";
    $bancodedados = "****";
    //testando a conexao com o servidor
    $ss = oci_connect($usuario1, $senha1, $servidor);
    //testando a conexao com o banco de dados
    if (!$ss) {
        $e = oci_error();
        trigger_error(htmlentities($e['Nao deu certo'],
           ENT_QUOTES), E_USER_ERROR);
    }
    $stid = oci_parse($ss, $sql);
    oci_execute($stid);

    $data = array();
    while ($row = oci_fetch_assoc($stid))
        array_push($data,$row);

    header('Content-Type: application/json');
    echo json_encode($data);

    oci_close($ss);
}
?>
```

Algoritmo C.1 – fetchdata

O código inicia com uma verificação para determinar se o parâmetro passado através da requisição GET não é nula.Em seguida, são definidas as variáveis para a conexão com o banco de dados Oracle, incluindo o nome do servidor, o nome de usuário, a senha e o nome do banco de dados. É feita a conexão com o servidor Oracle.

A consulta SQL é obtida do parâmetro "txtnome"e armazenada na variável $sql. Em seguida, é feita a preparação da consulta e sua execução. Os resultados da consulta são recuperados e são armazenados em um array chamado $data.

O tipo de conteúdo da resposta é definido como JSON e os resultados são codificados e enviados de volta para o *front-end*. Por fim, a conexão com o servidor Oracle é fechada garantindo que os recursos do servidor sejam liberados adequadamente após a conclusão da consulta.