Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Sistemas de Informação
Trabalho de Conclusão de Curso – Artigo

USO DE TÉCNICAS DE TUNING PARA GERAÇÃO DE DADOS EM DASHBOARDS

Fernanda Rebelatto Miranda[1]
Centro Universitário Academia, Juiz de Fora, MG
Evaldo de Oliveira da Silva[2]
Centro Universitário Academia, Juiz de Fora, MG

Linha de Pesquisa: Banco de Dados

RESUMO

Este artigo é um estudo de caso de um sistema real demonstrado com geração de *dashboards* de forma otimizada a partir de aplicação de técnicas de *tuning* em processo de ETL. Os dados utilizados são da base de dados do sistema Tetris oDash da empresa Tetris Solutions LTDA que possui um volume considerável e gerou a necessidade do desenvolvimento deste trabalho. São descritos conceitos de ETL, formas de aplicação de *tuning* como criação de índices nas tabelas do banco de dados; uso de *view* materializada; criação de consultas mais eficientes, e demonstração de melhoria no tempo de resposta de consulta após a aplicação de *tuning* na base de dados em questão. Para demonstração da criação de painéis com os relatórios gerados no Tetris oDash foi utilizado o *software* Metabase tornando possível a análise explicada de performance do banco de dados.

Palavras-chave: Tuning, ETL, Metabase.

[1] Discente do Curso de Sistemas de Informação do Centro Universitário Academia – UniAcademia. Endereço: R Halfeld, 1179, Centro. Celular: (32) 99981-0416. E-mail: fernanda.rebelatto@gmail.com
[2] Docente do Curso de Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora. Orientador.

1 INTRODUÇÃO

Este artigo é fruto de um relato de estudo de caso experienciado pela autora que é desenvolvedora do sistema Tetris oDash da empresa Tetris Solutions LTDA, respaudada pela autorização da empresa vide Anexo A.

Um dos problemas típicos no desenvolvimento de *software* é a mudança de escopo no projeto seguido de falta de controle dessas mudanças. Isso ocorreu no desenvolvimento do sistema de *dashboards* Tetris oDash, que inicialmente projetado para análise de seguidores e interações nas redes sociais do cliente, hoje entrega relatórios complexos de *marketing* (TETRIS, 2020).

Tetris oDash é uma ferramenta que ajuda na gerência de investimentos em campanhas de *marketing* e seus resultados coletando de forma automatizada, organizada e padronizada dados de diferentes fontes apresentando relatórios concisos em um único painel, mesmo que esses dados sejam de agências de publicidade diferentes, tendo como público alvo empresas com um grande volume de campanhas e/ou várias agências contratadas (TETRIS, 2020).

Assim como os objetivos do sistema Tetris oDash, a sua forma de armazenamento de dados também mudou nos últimos 5 anos. Com relatórios pré-definidos, sem opções de mudança de parâmetros dos mesmos e poucas métricas, todos os dias era gerado um JSON[3] (em inglês *JavaScript Object Notation*), um modelo simples de informações em formato de texto, que era consumido pelos *dashboards*. Mas com as métricas de *marketing* e flexibilização dos relatórios, como parâmetros livres de períodos de dados, optou-se pela criação de um banco de dados relacional de código aberto, o MySQL (MYSQL, 2020), para as consultas dos relatórios por ser um dos mais populares (TETRIS, 2020).

Os dados inseridos na base de dados do Tetris oDash advêm de diversas APIs (em inglês *Application Programming Interface*), que são conexões a outros aplicativos de *software*, de forma automatizada; carregamento de CSV (em inglês *Comma-separated values*), arquivo de texto que separa valores com vírgulas e pode

[3] https://www.json.org/json-pt.html

ser lido em formato de planilha; ou são inseridos manualmente através de um formulário (TETRIS, 2020).

Com o passar do tempo novos conceitos de *marketing* se desenvolveram e novas métricas foram criadas. Além disso, as APIs aumentaram o universo de dados disponíveis, entregando informações mais precisas, segmentadas. Esses dois fatores geraram demanda de adequação do Tetris oDash com o cenário atual para que o sistema não perdesse relevância no mercado. Essa adequação fez com que a tabela da base de dados que armazena os dados para consultas de relatórios aumentasse tanto em número de colunas quanto em número de registros diários, estes que antes somavam em média 8 mil, passaram a ser 50 mil registros novos por dia, oferecendo relatórios cada vez mais flexíveis e complexos, com várias informações parametrizadas. Porém, dependendo dos parâmetros escolhidos pelo usuário, as consultas à base de dados passaram a ter um tempo de resposta bem demorado, deixando o usuário esperando pelo seu relatório, sendo notável a perda de performance do sistema e a necessidade de otimização da base de dados que é o objetivo desse projeto de conclusão de graduação (TETRIS, 2020).

É preciso tomar boas decisões, levando em consideração o volume esperado de dados em cada relação do sistema, assim como as consultas que serão realizadas com mais frequência, no início, na fase de projeto do sistema de banco de dados, para que o mesmo tenha um bom desempenho. Mas, na maioria das vezes, apenas quando o sistema está em uso é que se percebe seu real desempenho e pode ser que as considerações dos projetistas mostram-se incorretas. Por isso, é comum a necessidade de ajuste no projeto, com base no seu comportamento real, para melhorar o desempenho e a estabilidade. Esta fase subsequente de ajuste chamamos de *tuning* (CARNEIRO, MOREIRA e FREITAS, 2020).

De acordo com Proni (2020), o MySQL é um banco de dados muito sensível a parâmetros de configuração e por isso uma simples alteração no seu arquivo de inicialização pode trazer grande benefício para a aplicação, apesar de não ter um bom desempenho com a configuração padrão.

Por esse motivo este artigo tem por objetivo a aplicação de técnicas de

tuning, para melhorar a interação do sistema com o banco de dados, assim como diminuir o tempo de resposta das consultas, otimizando o seu desempenho.

Os testes de desempenho que serão apresentados foram obtidos com um *notebook* de processador Intel Core i7, com 16GB de RAM, rodando sistema operacional Windows 10 Home Single Language (64 bits), e a versão adotada do MySQL foi a distribuição 5.5.5-10.4.11-MariaDB.

O restante do artigo segue organizado em seções a serem definidas. A Seção 2 descreve os conceitos de ETL (*Extract, Transform and Load*) e *tuning*, métodos utilizados para o desenvolvimento da proposta deste trabalho. A Seção 3 apresenta a aplicação das técnicas de *tuning* em processo de ETL. A Seção 4 descreve a geração de *dashboards* a partir das técnicas de *tuning* em ETL. Finalmente, a Seção 5 apresenta as considerações finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesta seção serão descritos os conceitos que fundamentam este trabalho, as técnicas, procedimentos e implementação da solução proposta.

2.1 PROCESSO DE ETL (*EXTRACT, TRANSFORM AND LOAD*)

O processo de ETL (*Extract, Transform and Load*) ocorre em 3 etapas: extração, transformação e carga dos dados de uma ou mais bases de dados de origem para uma ou mais bases de dados de destino (ABREU, 2008).

Quando o projeto consolida dados de diferentes fontes, como ocorre no Tetris oDash, o ETL torna-se indispensável, pois é esse processo que possibilita o uso sistematizado dos dados a partir da estratégia criada, garantindo a qualidade com que os dados em sua forma original são transformados em informação útil, legível e confiável, solucionando problemas de maior complexidade (KIMBALL, 2013).

Existem muitas formas de extrair dados, por exemplo, por uma API, ou por acesso a um banco de dados, ou por um arquivo. E justamente por isso esses dados que trazem as informações que precisamos para a carga na nossa base de destino

podem vir em formatos diferentes entre si, sem um padrão (KIMBALL, 2013).

Então passamos para a etapa de transformação do dado, unificando, padronizando essas informações, mas não só isso. Nessa etapa também aplicamos regras de negócio, tradução de dados codificados, ou a codificação de dados livres, cálculos, junção de vários dados em uma coluna só, ou a separação de um dados em várias colunas, são exemplos de transformações que podem ser feitas (KIMBALL, 2013).

Apesar das inúmeras possibilidades de transformação dos dados extraídos, essa é a única etapa do processo que é opcional. Podemos apenas extrair o dado e carregá-lo na base de dados de destino sem fazer modificação do mesmo (ABREU, 2008).

Por último, com os dados extraídos, modificados ou não, chegamos a etapa de carga na base de destino para que esses dados possam estar organizados, mapeados e disponíveis para serem consultados (KIMBALL, 2013).

2.2 TÉCNICAS DE *TUNING*

Um servidor de banco de dados precisa ser configurado de acordo com as necessidades e usos do sistema em questão para ser o mais eficiente possível. Essa configuração serve para ter o melhor aproveitamento do *hardware*, além de otimizar o servidor para realização das tarefas que demandam mais processamento ou que tratam de maior volume de dados (TELLES, 2020).

Quando um sistema entra em operação, o uso real das aplicações, transações, consultas e visões, revela fatores e áreas de problemas que podem não ter sido considerados durante o desenvolvimento do projeto inicial. Sendo assim, essas informações podem ser revisadas por meio da coleta de estatísticas reais sobre os padrões de uso para aplicação de técnicas de *tuning*. *Tuning* visa otimizar o desempenho na recuperação ou atualização de dados, minimizando o tempo de resposta de consultas/transações dos dados das aplicações, melhorando o desempenho geral das transações (CARNEIRO, MOREIRA e FREITAS, 2020).

Prado (2014) cita que existem muitas formas de aplicar técnicas de *tuning* que vão desde o sistema operacional até o código fonte e o processo de realização desse trabalho deve se dar por entender o problema, elaborar um diagnóstico e aplicar as técnicas que se aplicam a esse diagnóstico elaborado. Prado (2014)

também destaca que não existe uma regra que possa ser aplicada em qualquer caso, para qualquer banco de dados e que a experiência conduzirá o profissional ao uso das dicas e técnicas adequadas a cada problema encontrado. Dito isso, as técnicas aplicadas relacionadas ao banco de dados, segundo ele, podem ser descritas em três principais tipos de atividades:

- **Planejamento de performance**: Definição e configuração do ambiente em que o banco de dados será instalado, considerando-se os seguintes itens: *hardware*, *software*, sistema operacional e infraestrutura de rede.
- ***Tuning* de instância e banco de dados**: Ajuste de parâmetros e configurações do banco de dados (atividades que fazem parte do trabalho de um DBA, em inglês *Database Administrator*).
- **SQL *Tuning***: Basicamente significa realizar otimizações de instruções SQL, em busca de melhor desempenho.

Ou seja, podemos solucionar problemas de performance através de configuração de servidor, customizações das variáveis da base de dados, ou resolvendo consultas mal construídas.

Para ajudar no diagnóstico, verificar ou analisar as tabelas no metadados de um banco de dados MySQL em execução, a fim de sofrerem manutenção e *tuning*, pode-se usar o comando “SHOW VARIABLES;” para listar variáveis do sistema, conforme mostrado na Figura 01, e para ter acesso a estatísticas e indicadores de status o comando é “SHOW STATUS;”, conforme mostrado na Figura 02 (MYSQL, 2020).

Figura 01: Algumas variáveis listadas pelo comando SHOW VARIABLES

SHOW VARIABLES

Variable_name	
alter_algorithm	DEFAULT
analyze_sample_percentage	100.000000
aria_block_size	8192
aria_checkpoint_interval	30
aria_checkpoint_log_activity	1048576
aria_encrypt_tables	OFF
aria_force_start_after_recovery_failures	0
aria_group_commit	none

Fonte: Do autor (2020)

Figura 02: Algumas variáveis listadas pelo comando SHOW STATUS

SHOW STATUS

Variable_name	
Aborted_clients	1
Aborted_connects	1
Aborted_connects_preauth	0
Access_denied_errors	0
Acl_column_grants	0
Acl_database_grants	3

Fonte: Do autor (2020)

A seguir serão descritos como podemos melhorar o desempenho do banco de dados por meio de criação de índices nas tabelas e uso de *views* materializadas.

2.2.1 Criação de Índices

Existem muitas técnicas que podemos aplicar, mas na maioria das vezes, a primeira coisa a ser feita para acelerar consultas ao banco de dados é a criação de índices. Essa técnica é uma das mais importantes porque costuma fazer real diferença e se não aplicada de início, pode-se perder tempo tentando melhorar o desempenho através de outros meios (DUBOIS, 2007).

Porém mesmo agilizando as consultas, indexação deve ser usada com cautela, como explica Duarte (2008):

> O índice é uma estrutura que agiliza consideravelmente a busca pelos dados, mas introduz uma sobrecarga para as operações de escrita. Isto se deve ao fato de que a cada inserção, exclusão ou atualização de dados o MySQL terá que organizar a estrutura de índices, mantendo-os atualizados em relação aos dados. Assim, não se deve criar índices para colunas que não são utilizadas nas cláusulas WHERE e/ou GROUP BY. Além disto, antes da criação de um índice deve-se avaliar a seletividade da coluna a fim de se determinar se é interessante a criação de um índice nesta coluna. A seletividade do índice pode ser calculada dividindo-se o número de registros distintos na coluna pelo total de registros existentes na tabela, lembrando que quanto maior o número encontrado, maior será a capacidade seletiva desta coluna. Para ilustrar esta questão, uma coluna contendo apenas valores booleanos (true e false) não apresenta boa seletividade, visto que a quantidade de valores repetidos é muito grande em relação ao total de registros existentes (na média 50% são true e 50% são false). A questão da seletividade do índice é de extrema importância uma vez que o MySQL não utiliza um índice caso tenha que ler mais que 30% de toda a tabela. (DUARTE, 2008)

2.2.2 *View* Materializada

View é um resultado de consulta que fica armazenado no banco de dados e pode ser acessado como se fosse uma tabela e tem como principal função o controle de acesso de usuários a dados nas tabelas. Um usuário pode ter acesso a uma *view* e não a sua tabela de origem. *View* também pode ser usada para facilitar consultas complexas que porque já fica pré determinada e mostra os dados de forma mais organizada. Já *view* materializada é uma tabela de fato, que reflete o resultado da consulta do que seria uma *view* comum (ORACLE, 2020).

Por se tratar de uma tabela real no banco de dados a uma *materialized view* pode conter índices, como qualquer tabela, tornando a consulta ainda mais rápida, mas para apresentar dados corretos, atuais, essa tabela precisa ser atualizada sempre que ocorrer uma atualização em alguma tabela usada pela na consulta da sua construção. Ou seja, enquanto uma *view* comum realiza a consulta no momento que o usuário faz uma consulta, uma *materialized view* realiza a consulta no momento em que uma das tabelas consultadas é atualizada. Isso deve ser levado em consideração na hora de escolher criar uma *view* ou uma *materialized view* (ALVES, 2020).

Na prática porém, o MySQL não implementa *views* materializadas de forma nativa. Para simular seu funcionamento é preciso optar por adicionar *triggers* na tabela de origem para quando houver alguma modificação, seja inserção, alteração ou exclusão de registro, a *view* materializada correspondente seja atualizada também; e/ou adicionar rotinas de atualização da mesma (MYSQL, 2020).

Na próxima seção serão descritas como algumas técnicas de *tuning* foram aplicadas no Tetris oDash para melhorar sua performance.

3 APLICAÇÃO DAS TÉCNICAS DE *TUNING* EM PROCESSO DE ETL

A seguir serão listadas e descritas algumas técnicas de *tuning* que foram aplicadas pela autora deste artigo, desenvolvedora da Tetris Solutions LTDA, no ano de 2020, na base de dados do sistema Tetris oDash para melhoria de performance

(TETRIS, 2020).

3.1 REVISÃO DOS *TYPES* DAS COLUNAS

Colunas que eram DATETIME, mas que só eram usadas e convertidas nas consultas como DATE foram alteradas definitivamente para DATE, por exemplo a coluna “data_inicio” que foi alterada com o comando “ALTER TABLE dash_resultado_campanha MODIFY COLUMN data_inicio DATE” (TETRIS, 2020).

Colunas VARCHAR com tamanho maior que o necessário foram reduzidas, como por exemplo a coluna “creative_name” que era VARCHAR(1024) e foi alterada para VARCHAR(255) com o comando “ALTER TABLE dash_resultado_campanha MODIFY COLUMN creative_name VARCHAR(255)” (TETRIS, 2020).

3.2 CRIAÇÃO DE ÍNDICES EM COLUNAS USADAS EM “WHERE” NAS CONSULTAS

Alguns índices já existiam, mas se forem feitas muitas consultas utilizando colunas que ainda não possuem índice, a consulta fica lenta do mesmo jeito (TETRIS, 2020).

Além de criar índices comuns, também foram criados índices em múltiplas colunas, tornando as queries ainda mais eficaz, como por exemplo com o comando “CREATE INDEX idx_creative_data on dash_resultado_campanha (creative_name, data_inicio)” (TETRIS, 2020).

3.3 REGRA DE NEGÓCIO NA TRANSFORMAÇÃO DE DADOS ANTES DE FAZER A CARGA NO BANCO DE DADOS

Execução de regra ETL no código do sistema para unificar dados das colunas de geolocalização antes de salvar a informação na base de destino, fazendo não ser necessário nenhuma conversão de dados nas consultas (TETRIS, 2020).

3.4 *QUERIES* MAIS EFICIENTES

Anteriormente o sistema fazia uso de muitas conversões nas colunas para comparação de dados nas consultas, como por exemplo o uso de “DATE ()” para colunas do tipo DATETIME, e “UPPER()” para comparação de textos. O uso dessas conversões foi eliminado do sistema pois as colunas DATETIME agora são do tipo

DATE e não havia necessidade do uso de “UPPER()”, que transforma os textos em escrita com todas as letras maiúsculas porque se tratam de tabelas *case insensitive*, ou seja, não são sensíveis a texto com maiúsculas e minúsculas (TETRIS, 2020).

3.5 *VIEW* MATERIALIZADA

Como o número de colunas, segmentação das métricas de *marketing*, aumentaram, como por exemplo quantas visualizações específicas de um anúncio na *web*, o número de registros também aumentou muito, mas a maioria dos relatórios dizem respeito apenas ao total das métricas com um agrupamento de dados mais generalizado, por exemplo por plataforma, ou por marca. Dessa forma, para agilizar as consultas para a entrega de relatórios aos usuários, foi implementado o conceito de *view* materializada agrupando as métricas pelo segmentos mais requeridos do sistema (TETRIS, 2020).

Para isso foi criada uma tabela real, a “dash_resultado_campanha_mv”, para servir como a nossa *view* materializada com as colunas que foram definidas como as mais importantes, mais acessadas, para os relatórios do sistema Tetris oDash. Depois os dados foram inseridos através de uma consulta que resultaria na *view* desejada para alimentar a tabela recém criada dessa forma: “INSERT INTO dash_resultado_campanha_mv SELECT NULL, campanha_id, data_inicio, id_fonte_de_dados, veiculo, conta_externa_id, conta_externa_nome, sum(custo_total), sum(impressoes), [...] FROM dash_resultado_campanha WHERE segment_type IS NULL OR segment_type='ADS' GROUP BY campanha_id, data_inicio, id_fonte_de_dados, veiculo, conta_externa_id, conta_externa_nome;” (TETRIS, 2020).

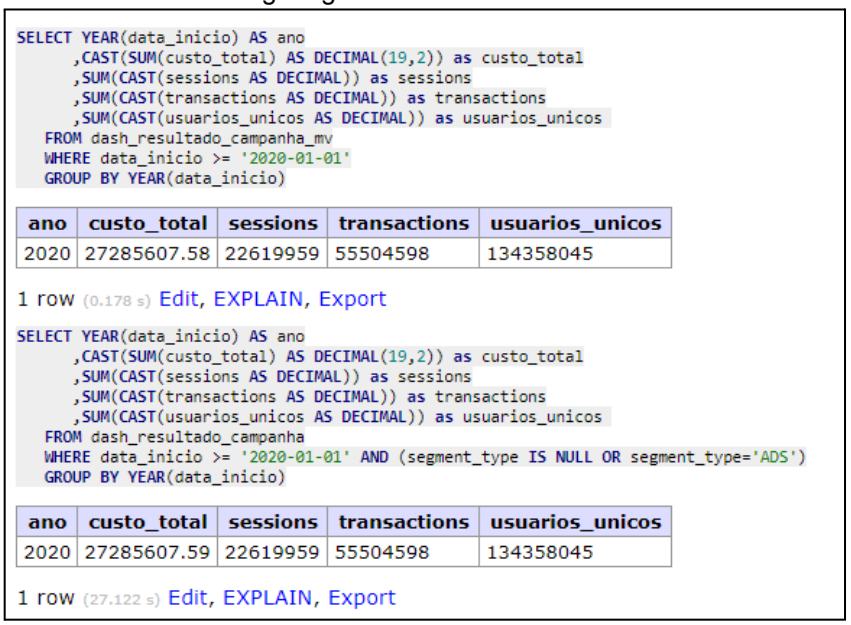
Como no MySQL a *view* materializada não se atualiza sozinha, foi criada também uma tabela de controle de registros alterados na tabela original que são inseridos por *triggers* disparados depois um qualquer tipo de alteração na tabela, seja CREATE, UPDATE ou DELETE. A essa tabela de controle demos o nome de “dash_resultado_campanha_mv_log” (TETRIS, 2020).

Sempre após a finalização de uma rotina de ETL, ou alteração manual da tabela de resultados, o sistema Tetris oDash dispara a PROCEDURE que passa para a *view* materializada as modificações que foram inseridas na tabela de controle e a tabela de controle é zerada, aguardando por novas mudanças (TETRIS, 2020).

A aplicação dessa técnica acelerou consideravelmente o sistema, transformando informações distribuídas em quase 14 milhões de registros em apenas 160 mil registros (TETRIS, 2020).

Na Figura 03 pode-se observar os resultados e tempo de consulta. A primeira com o uso da *view* materializada e a segunda acessando a tabela de resultados original.

Figura 03: Consultas de comparação de tempo de resposta entre uma view materializada e sua tabela de origem geradas com dados fictícios



```
SELECT YEAR(data_inicio) AS ano
      ,CAST(SUM(custo_total) AS DECIMAL(19,2)) as custo_total
      ,SUM(CAST(sessions AS DECIMAL)) as sessions
      ,SUM(CAST(transactions AS DECIMAL)) as transactions
      ,SUM(CAST(usuarios_unicos AS DECIMAL)) as usuarios_unicos
   FROM dash_resultado_campanha_mv
   WHERE data_inicio >= '2020-01-01'
   GROUP BY YEAR(data_inicio)
```

ano	custo_total	sessions	transactions	usuarios_unicos
2020	27285607.58	22619959	55504598	134358045

1 row (0.178 s) Edit, EXPLAIN, Export

```
SELECT YEAR(data_inicio) AS ano
      ,CAST(SUM(custo_total) AS DECIMAL(19,2)) as custo_total
      ,SUM(CAST(sessions AS DECIMAL)) as sessions
      ,SUM(CAST(transactions AS DECIMAL)) as transactions
      ,SUM(CAST(usuarios_unicos AS DECIMAL)) as usuarios_unicos
   FROM dash_resultado_campanha
   WHERE data_inicio >= '2020-01-01' AND (segment_type IS NULL OR segment_type='ADS')
   GROUP BY YEAR(data_inicio)
```

ano	custo_total	sessions	transactions	usuarios_unicos
2020	27285607.59	22619959	55504598	134358045

1 row (27.122 s) Edit, EXPLAIN, Export

Fonte: Do autor (2020)

Note que a única diferença na escrita da *query*, além da tabela FROM é a condição “AND (segment_type IS NULL OR segment_type='ADS')” necessária para a tabela de origem.

Enquanto a segunda consulta realizada na tabela de resultados original, demorou 27,122 segundos, a primeira entregou os resultados em 0,178 segundo, não deixando dúvidas das melhorias que o uso da *view* materializada trouxe ao Tetris oDash (TETRIS, 2020).

Na próxima seção será demonstrada a criação de painéis com o uso das *queries* otimizadas e análise explicada de performance do banco de dados.

4 GERAÇÃO DE *DASHBOARDS* A PARTIR DAS TÉCNICAS DE *TUNING* EM ETL

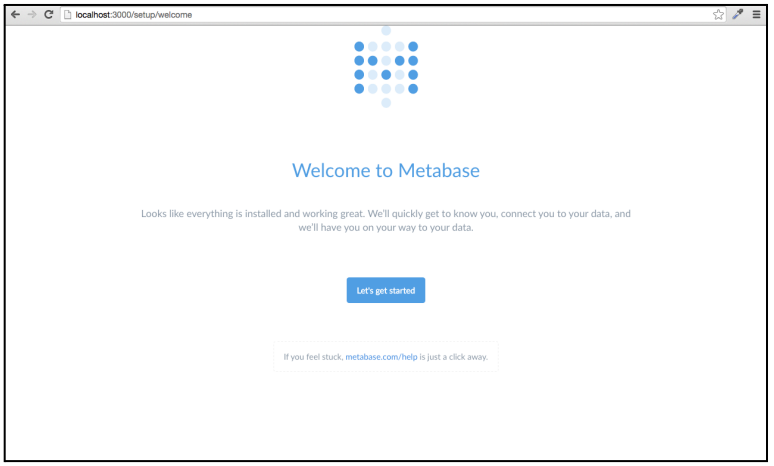
Para demonstrar a geração de *dashboards* com base nas técnicas de *tuning* aplicadas será utilizada a ferramenta Metabase. (METABASE, 2020)

A escolha por essa ferramenta se dá pela praticidade de instalação e configuração dos *dashboards*. (METABASE, 2020)

Para instalar o Metabase é necessário que você já tenha instalado o Java 8[4] ou versão superior. Se tiver, basta baixar o arquivo tipo "jar" presente no *link* <https://downloads.metabase.com/v0.35.4/metabase.jar> e executar o comando "java -jar metabase.jar" para iniciar o servidor Metabase na porta 3000, por padrão. (METABASE, 2020)

A Figura 04 mostra a tela de primeiro acesso ao Metabase instalado.

Figura 04: Tela inicial do Metabase

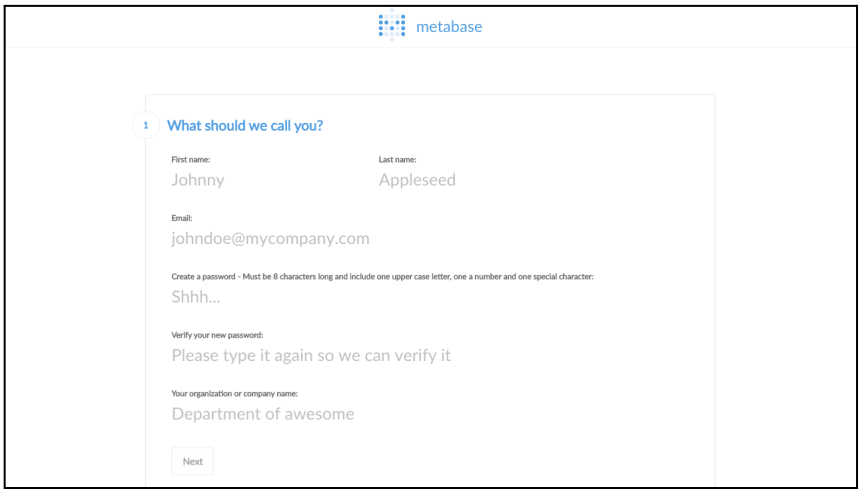


Fonte: METABASE (2020)

Após clicar em "*Let's get started*" você será encaminhado para um formulário de criação de conta de administrador, como é ilustrado na Figura 05. Essa conta, entre outros coisas, será usada para criar outros usuários se necessário e conectar o banco de dados.

Figura 05: Formulário de criação de conta de usuário

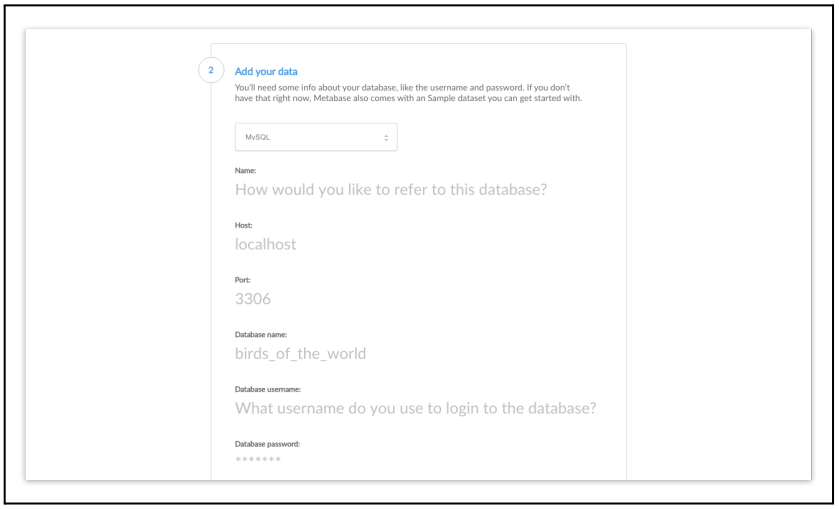
[4] https://www.java.com/pt_BR/download/



Fonte: METABASE (2020)

Com a conta de administrador criada, o próximo passo é conectar o banco de dados preenchendo o próximo formulário com os dados de acesso, como mostrado na Figura 06.

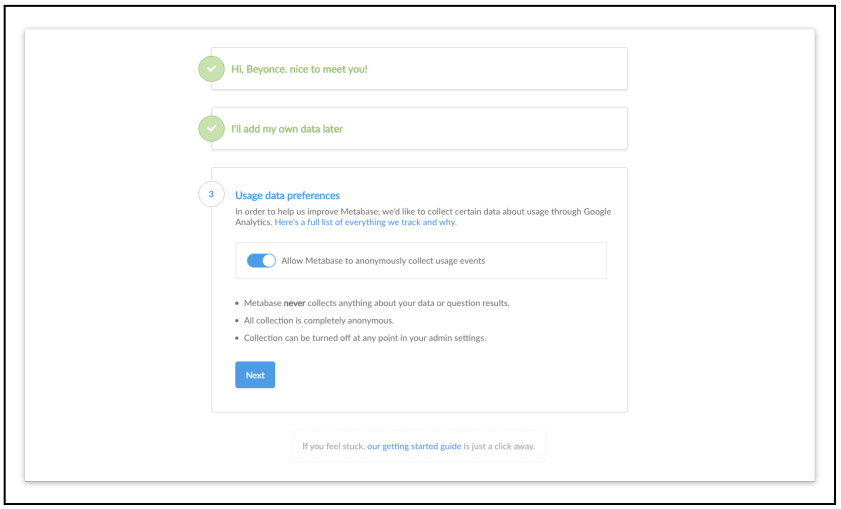
Figura 06: Formulário de conexão ao banco de dados



Fonte: METABASE (2020)

Ao final, como se pode ver na Figura 07, você é perguntado se o sistema pode coletar algumas informações anônimas sobre como você usa o Metabase para melhorar o seu desenvolvimento.

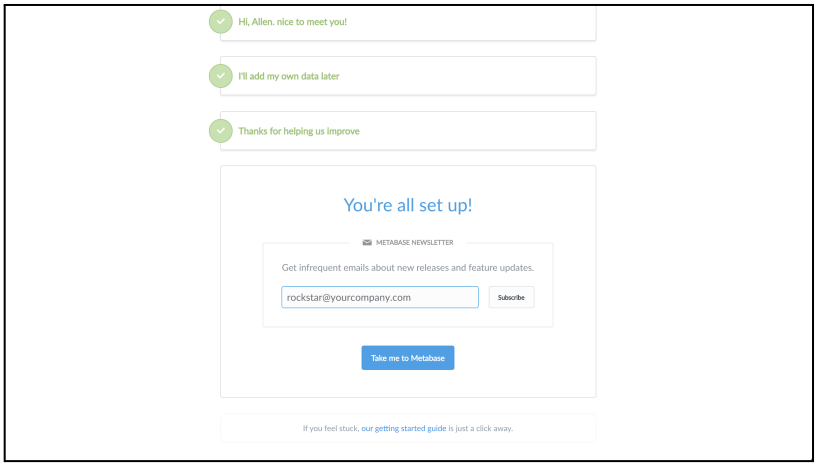
Figura 07: Formulário de coleta de dados anônimos



Fonte: METABASE (2020)

Você ainda tem a opção de assinar a newsletter do Metabase ou ir direto o uso da ferramenta clicando no link “Take me to Metabase”, demonstrado na Figura 08.

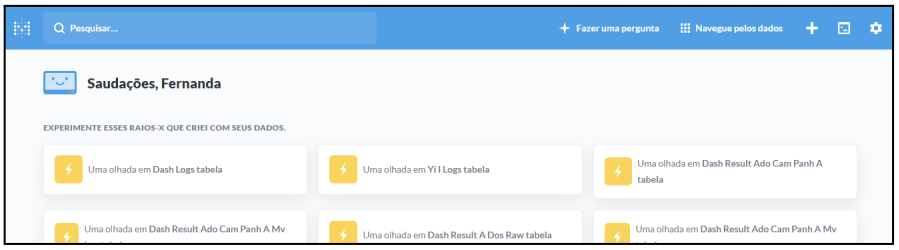
Figura 08: Formulário de newsletter



Fonte: METABASE (2020)

Depois do *login*, como pode-se ver na Figura 09, você já pode acessar as tabelas e ver análises geradas para elas. Mas o foco no momento é a criação os relatórios em *dashboards* e para isso você precisa clicar em “Fazer uma pergunta”.

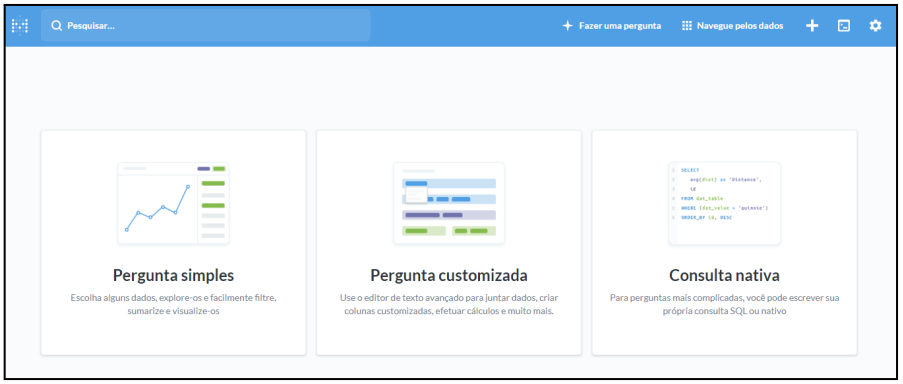
Figura 09: Tela inicial de acesso após login no Metabase



Fonte: Do autor (2020)

Dentre as opções para a criação de uma “pergunta” ao banco de dados, deve-se selecionar a “consulta nativa”, à direita como visto na Figura 10, para poder gerar uma consulta por *query* SQL.

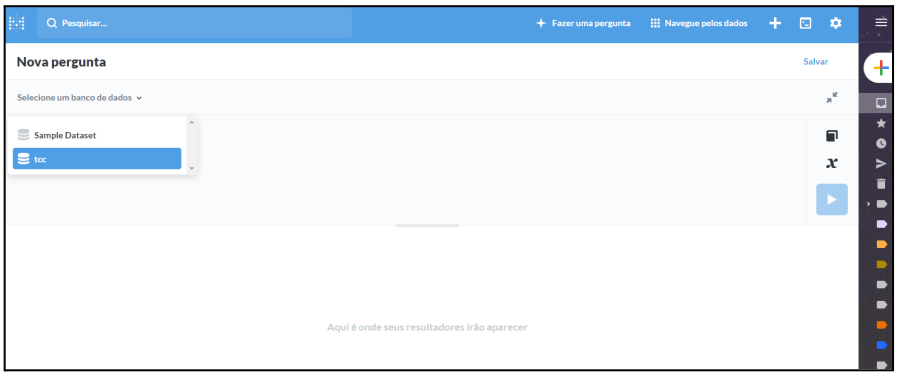
Figura 10: Seleção de tipo de nova pergunta no Metabase



Fonte: Do autor (2020)

Depois é preciso selecionar o banco de dados para poder escrever a *query* da consulta de forma otimizada que irá gerar o relatório. Pode-se ver essa tela na Figura 11.

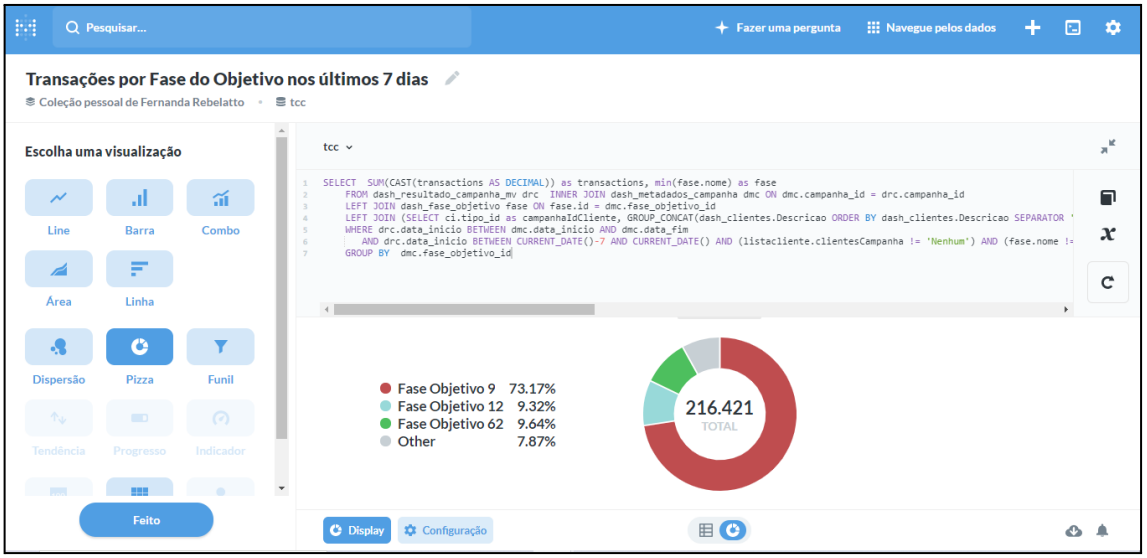
Figura 11: Seleção de banco de dados no Metabase



Fonte: Do autor (2020)

Depois da criação da consulta, pode-se selecionar um tipo de visualização do relatório que foi gerado podendo optar por uma tabela comum ou vários tipos de gráficos. Além disso você pode configurar esses gráficos, escolher título, colunas que irão representar a informação, cores entre outras coisas. Um exemplo de gráfico pode ser visto na Figura 12.

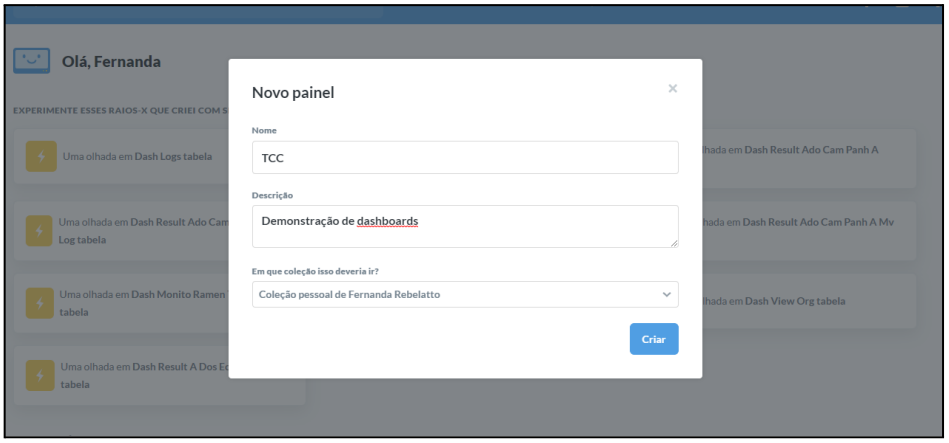
Figura 12: Gráfico tipo pizza gerado após consulta do banco de dados pelos Metabase



Fonte: Do autor (2020)

Com facilidade pela plataforma intuitiva e documentação a disposição bem ilustrada pode-se criar painéis com gráficos selecionando as “perguntas” já criadas ou criando novas a partir do próprio painel ativo. Na Figura 13 pode-se ver como se cria um novo painel. (METABASE, 2020)

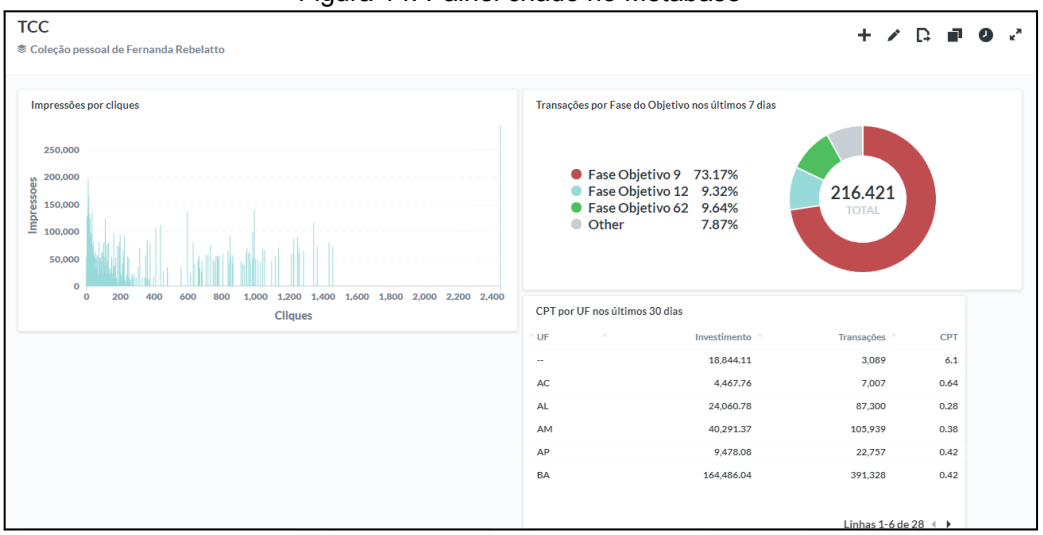
Figura 13: Criando novo painel no Metabase



Fonte: Do autor (2020)

Na Figura 14 pode-se ver um exemplo de painel criado com diferentes tipos de gráficos usados para análise de resultados.

Figura 14: Painel criado no Metabase



Fonte: Do autor (2020)

O Metabase é um gerador de *dashboard* que necessita de bancos de dados bem modelados e com acesso rápido para que as informações sejam recuperadas, senão seu funcionamento fica lento, conforme ocorre no sistema Tetris oDash. É importante que as consultas estejam otimizadas com o objetivo de diminuir o número de concorrências sob os registros (METABASE, 2020).

Essa ferramenta foi utilizada para testes das consultas e conclusão de que as otimizações aplicadas na Seção 3 deste artigo resolveram o problema de performance do Tetris oDash (TETRIS, 2020).

5 CONSIDERAÇÕES FINAIS

A otimização de banco de dados, assim como o próprio sistema a qual ele serve, vive em evolução e deve ser monitorado e analisado constantemente a fim de se evitar e/ou resolver problemas de performance, melhorando seu desempenho (RIBEIRO, 2010)

Ao desenvolver este projeto de conclusão de graduação foi possível perceber que não existe uma “receita de bolo” na hora de escolher quais técnicas de *tuning* aplicar para melhorar a performance de uma base de dados. Mas pode-se perceber que simples mudanças, como a criação de índices para as colunas das tabelas da base de dados usadas nas consultas condicionais, já fazem muita diferença. A decisão de qual técnica aplicar é específica para cada sistema e deve ser tomada com base nos dados coletados durante seu uso real.

Após os testes feitos com a ferramenta Metabase, apresentada na Seção 4, das técnicas de *tuning* aplicadas a base de dados do Tetris oDash, descritas na Seção 3 deste artigo, o sistema Tetris oDash melhorou seu desempenho diminuindo o tempo de resposta de consultas à sua base de dados, comprovando a eficiência de tais técnicas aplicadas e não deixando mais o usuário esperando pela geração e entrega de seus relatórios automatizados solicitados.

Graças a este trabalho foram acelerados relatórios importantes para um cliente, que não terá o nome ocultado por confidencialidade, da empresa Tetris Solutions LTDA que demoravam 3 minutos para serem entregues pelo sistema e agora demoram poucos segundos, fazendo com que o cliente não perca o interesse na contratação do Tetris oDash.

Como trabalho futuro, há o desejo de aplicar outras técnicas de *tuning*, como por exemplo o particionamento de tabela, para continuar otimizando a base de dados do sistema de *dashboards* Tetris oDash que a cada dia aumenta mais seu volume de dados.

ABSTRACT

This article is a case study of a real system demonstrated with the generation dashboards in an optimized way from the application of tuning techniques in the ETL process. The data used are from the database of the Tetris oDash system of the company Tetris Solutions LTDA, which has a considerable volume and generated the need for the development of this work. ETL concepts, ways of applying tuning such

as creating indexes in the database tables; use of materialized view; creation of more efficient queries, and demonstration of improved query response time after tuning the database in question. To demonstrate the creation of panels with the reports generated in Tetris oDash, the Metabase software was used, making it possible to explain the database's performance analysis.

Keywords: Tuning, ETL, Metabase.

REFERÊNCIAS

ABREU, Fábio Silva Gomes da Gama e. **Desmistificando o Conceito de ETL**. Revista de Sistemas de Informação, n°. 02 Jul./Dez. 2008. Disponível em: <http://www.fsma.edu.br/si/Artigos/V2_Artigo1.pdf>. Acesso em: 10 jun. 2020.

ALVES, Gustavo Furtado de Oliveira. **Qual a diferença entre View e Materialized View?** Disponível em: <https://dicasdeprogramacao.com.br/qual-a-diferenca-entre-view-e-materialized-view/>. Acesso em: 14 jun. 2020.

CARNEIRO, Alessandro Pinto; MOREIRA, Julinao Lucas; FREITAS, André Luis Castro de. **TUNING - Técnicas de Otimização de Banco de Dados** - Um Estudo Comparativo: Mysql e Postgresql. Disponível em: <http://repositorio.furg.br/bitstream/handle/1/1692/TUNING.pdf>. Acesso em: 10 jun. 2020.

DUARTE, Eber. **Otimização de consultas em MySQL**. Artigo da Revista SQL Magazine - Edição 48, 2008. Disponível em: <https://www.devmedia.com.br/artigo-sql-magazine-48-otimizacao-de-consultas-em-mysql/8127>. Acesso em: 14 jun. 2020.

DUBOIS, Paul. **Otimização de consultas no MySQL**. Artigo da Revista SQL Magazine - Edição 25, 2007. Disponível em: <https://www.devmedia.com.br/otimizacao-de-consultas-no-mysql/6178>. Acesso em: 14 jun. 2020.

KIMBALL, Ralph; ROSS, Margy. **The data warehouse toolkit:** The definitive guide to dimensional modeling. John Wiley & Sons, 2013.

METABASE. Matabase. Disponível em: <https://www.metabase.com/>. Acesso em: 10 jun. 2020.

MYSQL. MySQL Database. Disponível em: <https://www.mysql.com/>. Acesso em: 10 jun. 2020.

ORACLE. Create Materialized View. Database SQL Reference. Disponível em:<https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6002.htm>. Acesso em: 14 jun. 2020.

PRADO, Fábio. **Entenda o que é Tuning em banco de dados , os conceitos de**

Tuning e como praticar. 2014. Disponível em: <https://www.portalgsti.com.br/2014/05/Tuning-o-que-e.html>. Acesso em: 10 jun. 2020.

PRONI, Ricardo Portilho. **MySQL Performance Diagnostics & Tuning** - Parte 1. 2010. Disponível em: <https://www.devmedia.com.br/mysql-performance-diagnostics-tuning-parte-1/18424>. Acesso em: 10 jun. 2020.

RIBEIRO, Robson Afranio. **Performance do banco de dados MySQL**. 2010. Disponível em: <https://www.devmedia.com.br/performance-do-banco-de-dados-mysql/18508>. Acesso em: 14 jun. 2020.

TELLES, Marcelo Josué. **Tuning no MySQL**. 2011. Disponível em: <https://www.devmedia.com.br/tuning-no-mysql/23024>. Acesso em: 10 jun. 2020.

TETRIS. Tetris oDash. Disponível em: <http://tetris.co/produtos/odash/>. Acesso em: 20 jun. 2020.

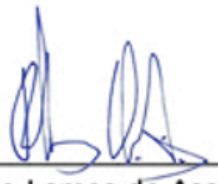
ANEXO A – DECLARAÇÃO DE AUTORIZAÇÃO DE USO

DECLARAÇÃO

Declaramos para os devidos fins, que Fernanda Rebelatto Miranda, inscrita no CPF nº 091.823.996-62, tem autorização para utilizar a estrutura da base de dados do sistema oDash com dados fictícios para uso exclusivo de seu projeto de TCC em 2020.

Local: Rio de Janeiro

Data: 10/06/2020



Pablo Lemos de Assis
Representante da Tetris Solutions LTDA
CNPJ 13.699.893/0001-57