Lógica de um CRM centralizada em operações PL SQL

Logic of a CRM centralized in PL SQL operations

Felipe Eduardo Neves de Oliveira
Graduando em Banco de dados pela Fatec Bauru
E-mail: felipe.eddy2@hotmail.com

Mateus Gomes Cabana
Graduando em Banco de dados pela Fatec Bauru
E-mail:Mateus.cabana@fatec.sp.gov.br

Renan Alves de Lira
Graduando em Banco de Dados pela Fatec Bauru
E-mail: renan.lira2@fatec.sp.gov.br

Profª. Drª. Patrícia Bellin Ribeiro
Docente na Fatec Bauru
E-mail: patricia.ribeiro5@fatec.sp.gov.br

RESUMO:

Com o passar do tempo as empresas começaram a investir em novas tecnologias através de implementações de sistemas de software baseados em micro-serviços que consiste em construir aplicações desmembrando-as em serviços independentes, visando escalabilidade e disponibilidade, deixando assim de ser puramente focadas em custos, para se tornarem organizações focadas em valor. Porém tal abordagem gera problemas como um aumento na complexidade no funcionamento, e consequentemente um aumento de I/O em banco gerando possíveis gargalos de indisponibilidade. Uma forma de se superar isso é concentrando toda a lógica da aplicação no banco, através de regras customizáveis como Triggers e Stored Procedures, a fim de evitar erros humanos. Este projeto desenvolveu um modelo de banco de dados relacional baseado em um de CRM (*Customer Relationship Management*) que utiliza a estrutura de micro-serviços em sua funcionamento, e centralizou todas as regra de negócio objetos do banco, como *Stored Procedures* e *Triggers* a fim de evitar possíveis de picos de indisponibilidade de transações "*lokantes".* Com isso se fez um estudo das diversas vantagens em se utilizar este tipo de abordagem, como a centralização das regras de negócio em apenas um ambiente, a facilidade do uso da linguagem PL/SQL, e a segurança da execução das regras pela atomicidade do sistema gerenciador de banco de dados, além de diminuir problemas com indisponibilidades e locks de aplicação.

Palavras-chave: Banco de Dados. CRM. Prospects. Clientes.

ABSTRACT:

Over time, companies began to invest in new technologies through the implementation of software systems based on micro-services, which consists of building applications by breaking them down into independent services, aiming at scalability and availability, thus no longer being purely focused on costs. , to become value-focused organizations. However, this approach generates problems such as an increase in the complexity of the operation, and consequently an increase in I/O in the bank, generating possible unavailability bottlenecks. One way to overcome this is to concentrate all application logic in the database, through customizable rules such as Triggerse Stored Procedures, in order to avoid human errors. This project developed a relational database model based on a CRM (Customer Relationship Management) that uses the micro-services structure in its operation, and centralized all business rules objects in the database, such as Stored Procedures and Triggers in order to avoid possible peaks of unavailability of “lokating” transactions. Thus, a study was made of the various advantages of using this type of approach, such as the centralization of business rules in just one environment, the ease of using the PL/SQL language, and the safety of rules execution due to the atomicity of the system database manager, in addition to reducing problems with outages and application locks.

Keywords: *Database. CRM. Prospect. Customer.*

1 INTRODUÇÃO

Com a melhoria de novas tecnologias as empresas estão deixando de ser puramente focadas em custos, para se tornarem organizações focadas em valor. Por isso, cada vez mais, as mesmas têm buscado maneiras de não apenas atender seus clientes (áreas de negócios), mas também de agregar valor no produto final e nos processos internos, através de implementações de sistemas de software baseados em arquiteturas de microsserviços, que consistem em construir aplicações desmembradas em serviços independentes, visando escalabilidade e disponibilidade. Por ser considerado um ativo estratégico esse modelo se tornou um grande diferencial, por gerar dados que possibilitam a tomada de decisão.

No universo de softwares, a forma mais tradicional de armazenar e manipular dados é a utilizando um banco de dados. Um banco de dados trata-se de uma coleção de dados armazenados e organizados, tendo em vista atender as necessidades do usuário, facilitando assim sua manipulação (PULGA et al., 2013). Ou seja, um banco de dados é um sistema que tem por finalidade registrar dados e permitir que seus usuários procurem e atualizem essas informações quando necessário (DATE, 2003).

É notório que com o advento de tais tecnologias foi necessário também adaptar regras de utilização, criar processos de manipulação de dados, ou encapsular lógicas de programação SQL (*Structured Query Language*) que permitam sua reutilização (Monteiro, 2018), visto que tornar tais tarefas manuais podem gerar erros humanos. Por isso, surgem mecanismos de controle que são chamados de *stored procedures* e *triggers*. Através destes é possível, por exemplo, tratar de dados de um cliente sem a necessidade de criar uma camada de abstração de software, que se comunica com o banco através de transações “*lockantes” (transações que devem ser finalizadas antes que outra possa ser executada)*, centralizando assim toda a lógica no contexto de banco de dados, visando manter a escalabilidade e disponibilidade da aplicação.

Sistemas que lidam com o gerenciamento de informações de clientes de uma empresa, são chamados de CRM (*Customer Relationship Management*), sigla em inglês para Gestão de Relacionamento com o Cliente. Utilizando este tipo de ferramenta as empresas têm maior controle de seus clientes, pois se tem acesso a todo o histórico de interações, compras, e outras informações relevantes para se obter métricas de desempenho.

Este trabalho tem como objetivo geral desenvolver um modelo de banco de dados de um sistema CRM que utilize arquitetura de microsserviços, e centralizar toda a regra de negócio no banco, utilizando *Stored Procedures* e *Triggers*. Com isso, será feita a análise da efetividade e integridade deste modelo adotado para sistemas que utilizam microsserviços.

Percebe-se que com o aumento da utilização de novas tecnologias, tornou-se necessário também mantê-las escaláveis, resilientes e acessíveis. Por isso, as empresas cada vez mais optam pela arquitetura de microsserviços. Porém, ela exige com que os programadores criem mecanismos na aplicação para manter a integridade desses dados, o que pode tornar a aplicação mais lenta devido à comunicação entre a aplicação e o banco ser uma comunicação *"lockante",* ou seja, não é possível executar nenhuma outra operação até que a resposta seja entregue. Vale ressaltar que validação de dados entre os serviços não é feita, o que gera *bugs* de produção e consequentemente prejuízos à empresa. O intuito deste trabalho é tentar apresentar uma solução para o problema de "lentidão" causada pelas transações *"lockantes",* e tentar aumentar a atomicidade e integridade dos dados, centralizando todas as ações no banco.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Banco de Dados

Um banco de dados é a forma tradicional de manipular e armazenar dados em coleção de dados, tendo em vista atender as necessidades do usuário final, ou seja, essas informações e dados devem estar organizados de forma consistente e sejam acessíveis de forma fácil e simples(PULGA et al, 2013), através da utilização da linguagem SQL.

Um banco de dados é controlado por um sistema gerenciador de banco de dados (SGBD), que junto com os aplicativos associados a ele formam um sistema de banco de dados (ORACLE, 2021). O SGBD serve como uma interface entre o banco de dados e os usuários, que podem ser pessoas ou mesmo *softwares*. Alguns exemplos de softwares de bancos de dados populares ou DBMSs incluem MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database e dBASE.

O modo como os dados são armazenados se assemelha a planilhas, como a do Microsoft Excel, mas existem diversas diferenças entre os dois. Como a permissão para acesso aos dados, e a quantidade de dados que podem ser armazenados. Planilhas são ótimas para um usuário ou um pequeno número de usuários que não precisam fazer manipulações de dados complicadas. Bancos de dados, por outro lado, são projetados para conter coleções muito maiores de informações organizadas, quantidades enormes, às vezes. Os bancos de dados permitem que vários usuários, ao mesmo tempo, acessem e consultem com rapidez e segurança os dados usando lógica e linguagem altamente complexas.

O que torna o processamento de dados eficiente, é a forma como estão estruturados, que é em linhas e colunas em diversas tabelas. Os dados,

armazenados em tabelas, são inseridos, atualizados e removidos através de uma linguagem de programação chamada SQL (*Structured Query Language*).

2.1.1 Structured Query Language - SQL

Structured Query Language ou SQL, é uma linguagem criada em 1986 pela IBM que é utilizada para a realizar a interação com o banco de dados (Plew E Stephens, 2000). Sendo uma das mais utilizadas para Bancos de Dados relacionais, essa linguagem permite realizar diversas operações como consultas, inserção, deleção e até atualização de dados.

A linguagem sql pode ser subdivida em três tipos de sub linguagens, *Data Definition Language* (DDL) que é responsável pela criação de objetos de dados como tabelas views entre outros, *Data Manipulation Language*(DML) são aquelas responsáveis pela manipulação de dados como selecionar, inserir, atualizar e deletar dados e *Data Control Language* (DCL) responsável por controlar o acesso dos dados e usuários que terão acesso aos mesmos (Júnior, 2008).

2.1.1.2 Sintaxe

O modelo SQL faz uso de tabelas para armazenar os dados. Nestas tabelas, colunas representam os campos e as linhas representam os registros em si. Já que é uma linguagem de modelos relacionais, é necessário que as colunas estejam associadas entre si. Essa associação se dá por meio das colunas, que fazem referência a dados presentes em outras tabelas. Dessa forma, sabe-se a qual registro aquela linha está originalmente ligada. A programação se tornou mais fácil e a quantidade de códigos a serem gerados diminuiu consideravelmente (MELTON,1994).

É uma linguagem de fácil aprendizado e que atende a todas as necessidades de manutenção e administração para a maior parte dos Bancos de Dados, fazendo com que tenha um alto nível de aceitação entre os usuários. A estrutura de seus comandos vêm do inglês, sendo de fácil intuição até mesmo para quem não é fluente no idioma.

A estrutura de seus comandos é basicamente sempre a mesma. Inicia-se com um verbo indicando a ação que deseja executar. Dentre as mais utilizadas estão: SELECT, INSERT, UPDATE e DELETE, sendo utilizadas para selecionar, inserir, atualizar e apagar registros, respectivamente. Depois da ação indicada pelo verbo, devem ser informados o(s) campo(s) e tabela(s) que devem ser afetados. A linguagem SQL ainda aceita alguns comandos de filtragem (WHERE, que indica uma condição que a busca deve atender), ordenação (ORDER BY, que ordena por ordem crescente/decrescente, por exemplo), agrupamento (GROUP BY, juntando resultados que apresentem uma mesma característica), entre outros. (Devmedia, 2021)

2.1.2 Banco de dados relacional

Um banco de dados como mencionado anteriormente organiza um modelo de dados. Segundo Silberschatz (2006) um modelo de dados é uma forma de explicar as características que descrevem os dados seus relacionamentos, relações de dados, semântica de dados e restrições. Um *banco de dados relacional* é um tipo de banco de dados que é baseado no modelo relacional, ou seja, consegue relacionar seus dados através de tabelas e registros, cada colunas da tabela contém atributos dos dados e cada registro geralmente tem um valor para cada atributo, facilitando o estabelecimento das relações entre os pontos de dados (Reis, 2008).

2.1.3 MySQL

O MySQL é um SGBD (Sistema Gerenciador de Banco de Dados) relacional que utiliza a linguagem SQL. Atualmente é um dos sistemas mais populares do mundo, utilizado em grande parte das aplicações gratuitas para gerir suas bases de dados. (Teixeira, 2013)

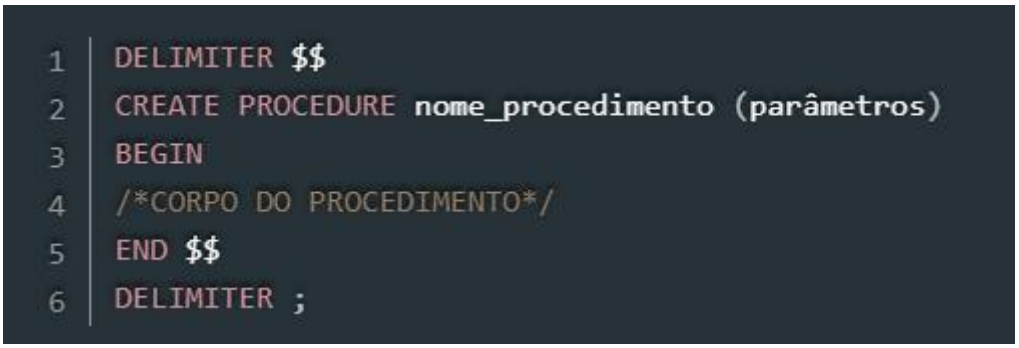
É uma excelente escolha pela sua portabilidade e compatibilidade com praticamente qualquer plataforma. Também possui um excelente desempenho e estabilidade sem exigir muito de recursos de *hardware*.

Dentre os muitos usuários do MySQL estão também grandes nomes, como: Banco Bradesco, HP, Nokia, Sony, U.S. Army, Google, NASA, entre outros (Mysql, 2021).

2.1.4 Stored Procedures

Diferentemente das triggers, as Stored Procedure são procedimentos armazenados que contém um conjunto de comandos SQL que podem ser executados como uma função, com isso ela possibilita parâmetros de entrada e saída. Ela pode ser utilizada para ser acionada através de uma chamada simples que executa uma série de outros comandos (Oracle, 2021). A imagem 1 descreve a sintaxe de uma procedure no Mysql.

Figura 1 - Sintaxe da Stored Procedure



Fonte: Rodrigues,2013

2.1.5 Triggers

Trigger, “gatilho” em inglês, é uma estrutura do banco de dados, uma função vinculada a uma tabela que é executada diante de alguma condição ou ação. Ou seja, sua execução depende de outra ação no banco, podendo ser antes ou depois, fazendo jus ao nome de gatilho. Por exemplo, uma trigger pode ser chamada antes de inserir dados em uma tabela para fazer verificações, ou pode ser chamada após a atualização de um registro para inserir um log em uma outra tabela. Vale destacar que uma trigger não pode ser chamada diretamente, diferentemente dos procedimentos armazenados no sistema (ORACLE, 2021). A imagem 2 demonstra a sintaxe de um trigger na linguagem Mysql.

Figura 2 - Sintaxe da Stored Procedure

```
CREATE TRIGGER <trigger name>
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON <table name>
  FOR EACH ROW
  <triggered action>
```

Fonte: Oracle, 2021

2.2 Arquiteturas de software

A arquitetura de software refere-se a como os componentes de uma aplicação relacionam-se e como se comportam em uma aplicação, sejam elas consideradas apenas um bloco ou pedaços independentes (FILHO, 2008), ou seja é um modelo sob o qual um sistema pode ser desenvolvido. Tendo em vista essa definição, a arquitetura de software pode ser dividida em dois tipos: arquitetura monolítica e arquitetura de microsserviços que serão melhor definidas no próximo tópico. Vale destacar que a escolha da arquitetura é um processo fundamental visto que ela possui características que podem influenciar na qualidade, performance e manutenção de um projeto, sendo decisivo para o sucesso da aplicação.

2.2.1 Arquitetura Monolítica

A arquitetura monolítica é definida como um grande serviço os quais todos os componentes de uma aplicação formam uma única unidade de implantação (Richardson, 2014), ou seja é feito em um sistema, que roda em um único processo, dessa forma seus diferentes componentes estão ligados em apenas um programa dentro de uma plataforma. Vale destacar que essa modelagem é frequentemente utilizada em muitas empresas devido a sua simplicidade de implementação e desenvolvimento, porém a medida que a complexidade das regras de negócios aumenta, pode acarretar na geração de muitos problemas como tamanho do código e dificuldade na sua manutenção, visto que os componentes possuem uma baixa modularidade, ou seja uma porção de código que realiza uma apenas uma tarefa específica, o que contribui para uma má qualidade do código ao longo do tempo. Vale ressaltar que a escalabilidade tende a ser um problema, devido ao fato que os diversos componentes têm necessidades diferentes de recursos e com isso geraria gargalos na aplicação pela dificuldade de escalar cada componente de forma independente, diferentemente da arquitetura de microsserviço.

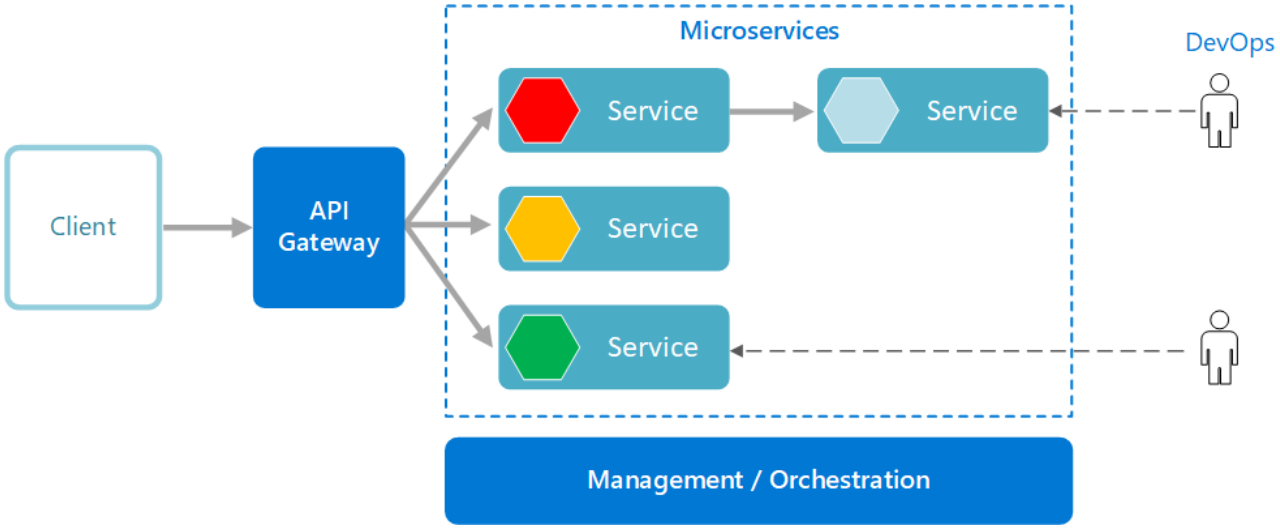
Figura 3 - Exemplo representativo de uma aplicação que utiliza arquitetura monolítica

Fonte: AWS, 2021

2.2.1 Arquiteturas de microsserviço

A arquitetura de microsserviços são aplicações independentes capazes de comunicar com outras partes de uma aplicação possuindo módulos isolados de persistência de dados, visando a escalabilidade (capacidade da aplicação crescer verticalmente ou horizontalmente conforme a necessidade) e a disponibilidade (é a habilidade de resposta da aplicação) (Dragoni et al. 2016). Vale ressaltar que por mais que seu nome seja micro, não necessariamente se trata de um sistema de baixa complexidade e acoplamento, ou pequeno. Os micro serviços podem ser grandes e complexos ligados a outros sistemas de diversos tamanhos.

Figura 4 - Exemplo de arquitetura de Microsserviços



Fonte: Microsoft, 2021

2.2.2 Locks blocks e deadlocks

Existem três tipos de transações. *Locks*, *blocks* e *deadlocks.* Os *Locks* impedem que os mesmos dados sejam atualizados simultaneamente por mais de um usuário, o que poderia abalar a integridade do Banco de Dados. *Block* é o que ocorre quando dois processos precisam ter acesso simultaneamente ao mesmo trecho de dados, de forma que ocorra um *Lock*. Assim que o primeiro processo terminar, o bloqueio é desfeito para que o fluxo seja continuado. *Deadlock* acontece quando um primeiro processo aguarda o término do segundo, mas o segundo em algum momento entra em *block* para aguardar o primeiro, que já está em *block.* Assim, nenhum dos dois é capaz de executar. Neste trabalho serão abordadas transações do tipo *Lock*. (Mulotto, 2019)

3 METODOLOGIA

Para a realização desta pesquisa, utilizou-se o banco de dados MySQL, por vários fatores, entre eles pode-se citar: ser um banco open source, o que facilita em nossa pesquisa em relação a orçamento de ferramentas. Um banco altamente disseminado auxiliando assim em busca de documentação e possui uma comunidade que contribui continuamente. Vale ressaltar que o mesmo tem integração com outros bancos de dados como db2 e Oracle, sendo esse, um dos principais pontos para a escolha do banco, visto que o banco de dados Oracle é o banco mais utilizado no mundo entre as empresas, porém é um banco pago, e escolhendo uma ferramenta que pode interfacear os resultados da pesquisa com o ambiente empresarial é um grande ponto positivo.

Apesar do MySQL possuir diversas ferramentas de gerenciamento de dados optou se por escolher o workbench, visto que ele possui ferramentas que auxiliarão todos os passos da pesquisa como a criação de um diagrama entidade relacionamento do modelo proposto, até a criação dos scripts para a integração do CRM.

Para a realização desta pesquisa utilizou-se um modelo Lógico, que possui a entidade pessoa, que terá o status de Prospect e Cliente, da entidade produto e pedido. Essa estrutura foi embasada em softwares CRM (Customer Relationship Management que são sistemas para Gestão de Relacionamento com o Cliente), os quais utilizam arquiteturas de microsserviços. Com base no modelo lógico, foi gerado o modelo físico para a realização do experimento e análise. Foram gerados dados de forma randômica

para alimentar a base de pessoas (uma amostra total, de 1000, 5000 e 10000). Por meio de Procedures as pessoas serão convertidas de “Prospects” para Clientes, e o estoque dos produtos será verificado antes de serem inseridos nos pedidos. As procedures serão executadas por triggers. Após as execuções serão feitas análises avaliando o grau de eficiência e principais vantagens de adotar esse método, visando manter a escalabilidade e integridade dos dados da aplicação. Após a execução das procedures e triggers será realizada uma análise dos resultados sobre a amostra, visando os seguintes pontos: todos a lógica o fluxo de venda foi feita, incluindo a atualização dos dados de clientes prospects sem a perda de nenhum dados, em caso de falhas qual o código retornado e com isso nenhum dado deve ser persistido. Além disso, será analisada qual a porcentagem de falhas e sucesso sob a amostra, visando avaliar o grau de eficiência do modelo proposto, bem como os pontos positivos e negativos.

Com o banco de dados criado e dados inseridos, serão feitas análises avaliando o grau de eficiência e principais vantagens de adotar esse método, visando manter a escalabilidade e integridade dos dados da aplicação. Após a execução das procedures e triggers será realizada uma análise dos resultados sobre a amostra, visando os seguintes pontos: se toda a lógica e o fluxo de venda foi executado com sucesso, incluindo a atualização de prospects em clientes sem a perda de nenhum dado. Em caso de falhas, analisaremos o registro de erros com o código retornado, para verificar qual registro não teve suas informações atualizadas por qual razão. Além disso, será analisada qual a porcentagem de falhas e sucesso sob a amostra, visando avaliar o grau de eficiência do modelo proposto, bem como os pontos positivos e negativos.

E com isso deseja-se obter dados sob a efetividade de um modelo de aplicação que utiliza microsserviços centralizando toda sua lógica nos objetos do banco de dados, com o propósito de reduzir o máximo possível as transações “*lockantes”*, e permitindo assim que a aplicação fique cada vez mais escalável e disponível.

4 RESULTADOS E DISCUSSÃO

A arquitetura de micro-serviços são aplicações independentes capazes de comunicar com outras partes de uma aplicação possuindo módulos isolados de persistência de dados, visando a escalabilidade (capacidade da aplicação crescer verticalmente ou horizontalmente conforme a necessidade) e a disponibilidade (é a habilidade de resposta da aplicação) (Dragoni et al. 2016). Vale ressaltar que com o desenvolvimento desses micro serviços, há um aumento na complexidade no funcionamento, visto que serão criados vários mecanismos de comunicação entre os processos e com isso diversas validações devem ser implantadas de modo a manter o funcionamento correto do sistema, um exemplo disso são as transações que passam a ser distribuídas, dificultando a escrita de testes automatizados para vários serviços, e criando diversos pontos de atrasos no código, devido às transações “lockantes” entre aplicação e banco.

Percebe-se que o modelo proposto no módulo anterior nos trás inúmeras vantagens. A primeira é que todos os passos foram desenvolvidos em uma linguagem SQL e com isso 7 não existe nenhuma ferramenta que precise ser adquirida do fornecedor, pois toda estrutura necessária já é integrada ao próprio SGBD (ou ao Sistema Operacional em alguns casos), a menos que se escolha por um SGBD pago. Além disso é uma linguagem de fácil aprendizado o que facilita na cooperação de outros especialistas para o produto desenvolvido, e sua similaridade facilita a integração do modelo nos diversos bancos de dados existentes. Em relação as procedures e triggers, elas permitem uma customização ao manuseio de dados sem restrições, e com isso as necessidades específicas são rapidamente atendidas. Vale destacar que por ser uma linguagem SQL a mesma é interpretada em tempo de execução, o que poderia ser um empecilho em um

ponto de vista comercial pois seria necessário criar um produto “fechado” e com isso agregar uma camada exclusiva para esse fim, além disso para regras extremamente complexas seria necessário um conhecimento maior sobre as engines do banco, visto que seria necessário criar mecanismos de otimização para que desempenho das tarefas seja o mais otimizadas possíveis não impactem no tempo de execução final. As tabelas abaixo mostram o grau de eficiência das execuções dos Jobs de testes.

Tabela 1 - Resultados do experimento com atualização de estoque e cliente.

Número de clientes atualizados por teste	**Número de pedidos realizado por cliente**	**Taxa de sucesso de atualização na tabela de cliente**	**Taxa de sucesso de atualização nas tabelas estoque e pedido**	**Taxa de sucesso na vendo do produto**
1000	5	100%	100%	100%
5000	10	100%	100%	100%
10000	20	100%	100%	100%

Fonte: Elaborado pelo Autor

Tabela 2 - Resultados do experimento com atualização de estoque com clientes efetivos.

Número de clientes atualizados por teste	**Número de pedidos realizado por cliente**	**Taxa de sucesso de atualização na tabela de cliente**	**Taxa de sucesso de atualização nas tabelas estoque e pedido**	**Taxa de sucesso na vendo do produto**
1000	5	0% (Clientes já eram efetivos)	100%	100%
5000	10	0% (Clientes já eram efetivos)	100%	100%
10000	20	0% (Clientes já eram efetivos)	100%	100%

Fonte: Elaborado pelo Autor

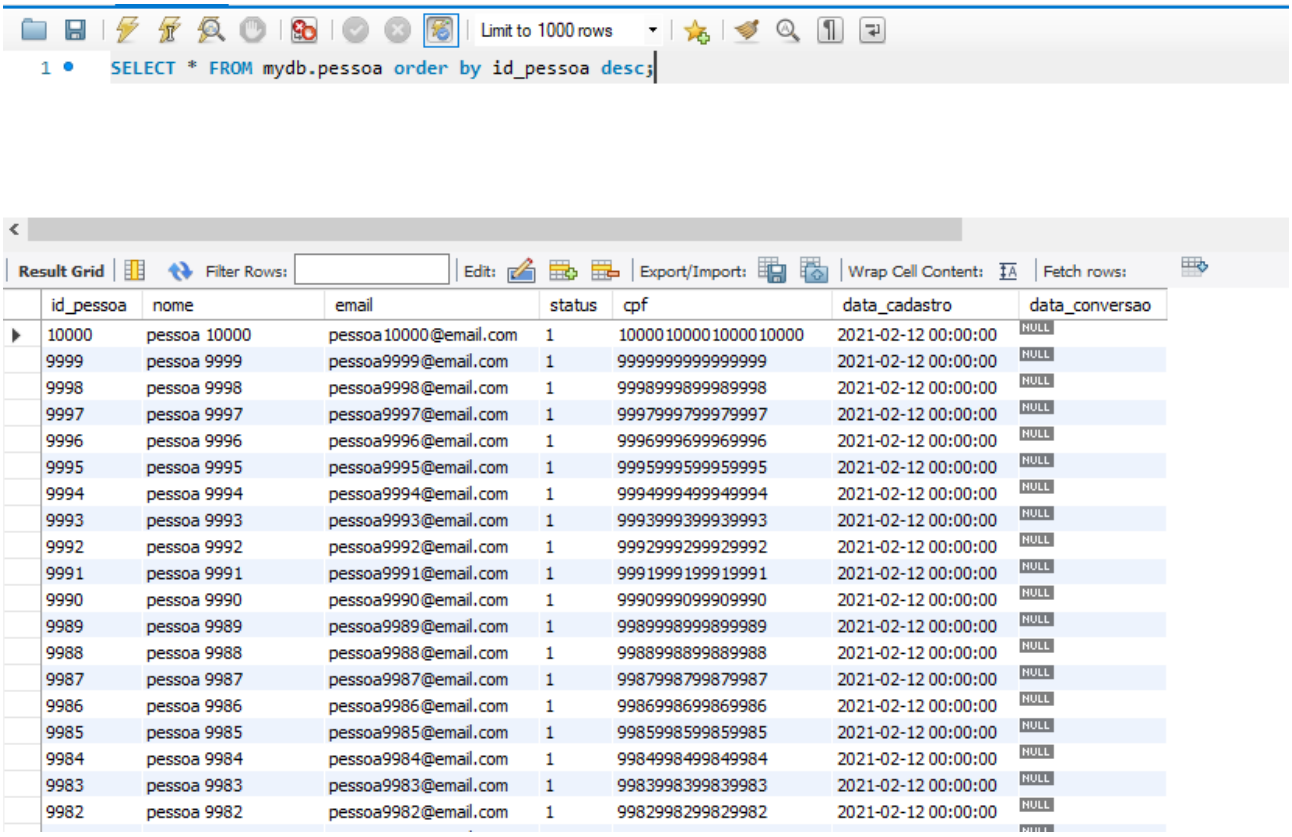
Tabela 3: Resultados do experimento de pedidos sem estoque de cliente prospects

Número de clientes atualizados por teste	Número de pedidos realizado por cliente	Taxa de sucesso de atualização na tabela de cliente	Taxa de sucesso de atualização nas tabelas estoque e pedido	Taxa de sucesso na venda do produto
1000	5	100%	0%(Não houve atualização)	0%(Estoque sem produto)
5000	10	100%	0%(Não houve atualização)	0%(Estoque sem produto)
1000	20	100%	0%(Não houve atualização)	0%(Estoque sem produto)

Fonte: Elaborado pelo Autor

As figuras 5,6 e 7 mostram as tabelas de cliente,pedidos e estoque, respectivamente antes das execução das triggers e procedimentos para o caso de 10000 clientes.

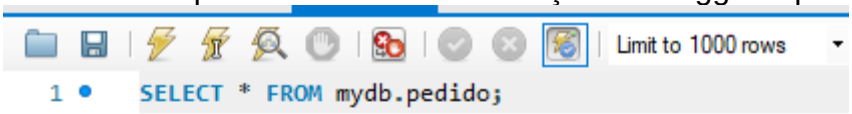
Figura 5 - Tabela de clientes prospect antes de serem clientes efetivos

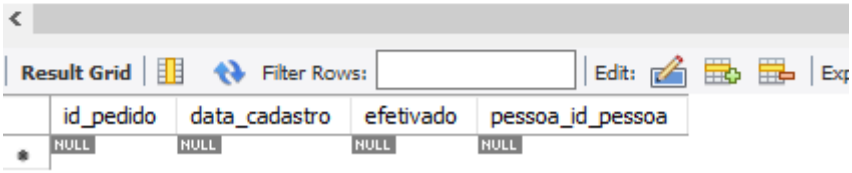


id_pessoa	nome	email	status	cpf	data_cadastro	data_conversao
10000	pessoa 10000	pessoa10000@email.com	1	10000100001000010000	2021-02-12 00:00:00	NULL
9999	pessoa 9999	pessoa9999@email.com	1	9999999999999999	2021-02-12 00:00:00	NULL
9998	pessoa 9998	pessoa9998@email.com	1	9998999899989998	2021-02-12 00:00:00	NULL
9997	pessoa 9997	pessoa9997@email.com	1	9997999799979997	2021-02-12 00:00:00	NULL
9996	pessoa 9996	pessoa9996@email.com	1	9996999699969996	2021-02-12 00:00:00	NULL
9995	pessoa 9995	pessoa9995@email.com	1	9995999599959995	2021-02-12 00:00:00	NULL
9994	pessoa 9994	pessoa9994@email.com	1	9994999499949994	2021-02-12 00:00:00	NULL
9993	pessoa 9993	pessoa9993@email.com	1	9993999399939993	2021-02-12 00:00:00	NULL
9992	pessoa 9992	pessoa9992@email.com	1	9992999299929992	2021-02-12 00:00:00	NULL
9991	pessoa 9991	pessoa9991@email.com	1	9991999199919991	2021-02-12 00:00:00	NULL
9990	pessoa 9990	pessoa9990@email.com	1	9990999099909990	2021-02-12 00:00:00	NULL
9989	pessoa 9989	pessoa9989@email.com	1	9989998999899989	2021-02-12 00:00:00	NULL
9988	pessoa 9988	pessoa9988@email.com	1	9988998899889988	2021-02-12 00:00:00	NULL
9987	pessoa 9987	pessoa9987@email.com	1	9987998799879987	2021-02-12 00:00:00	NULL
9986	pessoa 9986	pessoa9986@email.com	1	9986998699869986	2021-02-12 00:00:00	NULL
9985	pessoa 9985	pessoa9985@email.com	1	9985998599859985	2021-02-12 00:00:00	NULL
9984	pessoa 9984	pessoa9984@email.com	1	9984998499849984	2021-02-12 00:00:00	NULL
9983	pessoa 9983	pessoa9983@email.com	1	9983998399839983	2021-02-12 00:00:00	NULL
9982	pessoa 9982	pessoa9982@email.com	1	9982998299829982	2021-02-12 00:00:00	NULL

Fonte: Elaborado pelo Autor

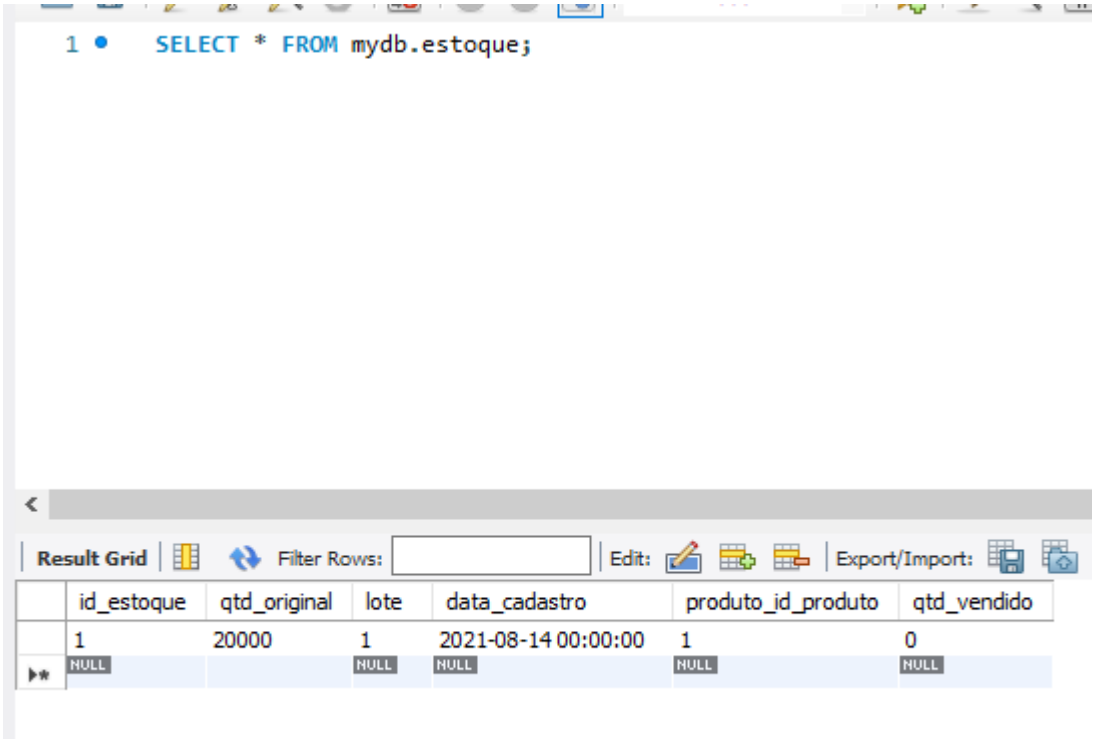
Figura 6 - Tabela de pedidos antes das execuções de trigger e procedures





Fonte: Elaborado pelo Autor

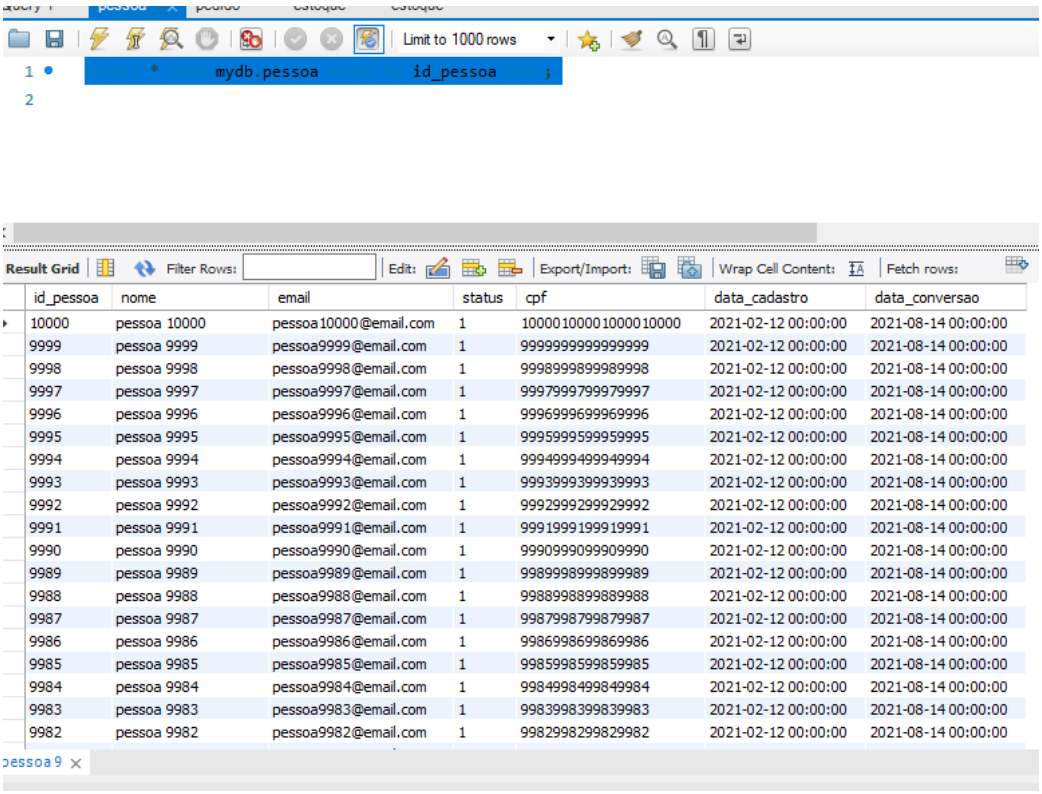
Figura 7 - Tabela de estoque antes das execuções de trigger e procedures



Fonte: Elaborado pelo Autor

Já as figuras 8,9 e 10 mostram os resultados finais das tabelas de cliente, pedido e estoque respectivamente após a execução das trigger e procedures para o mesmo número de amostragem.

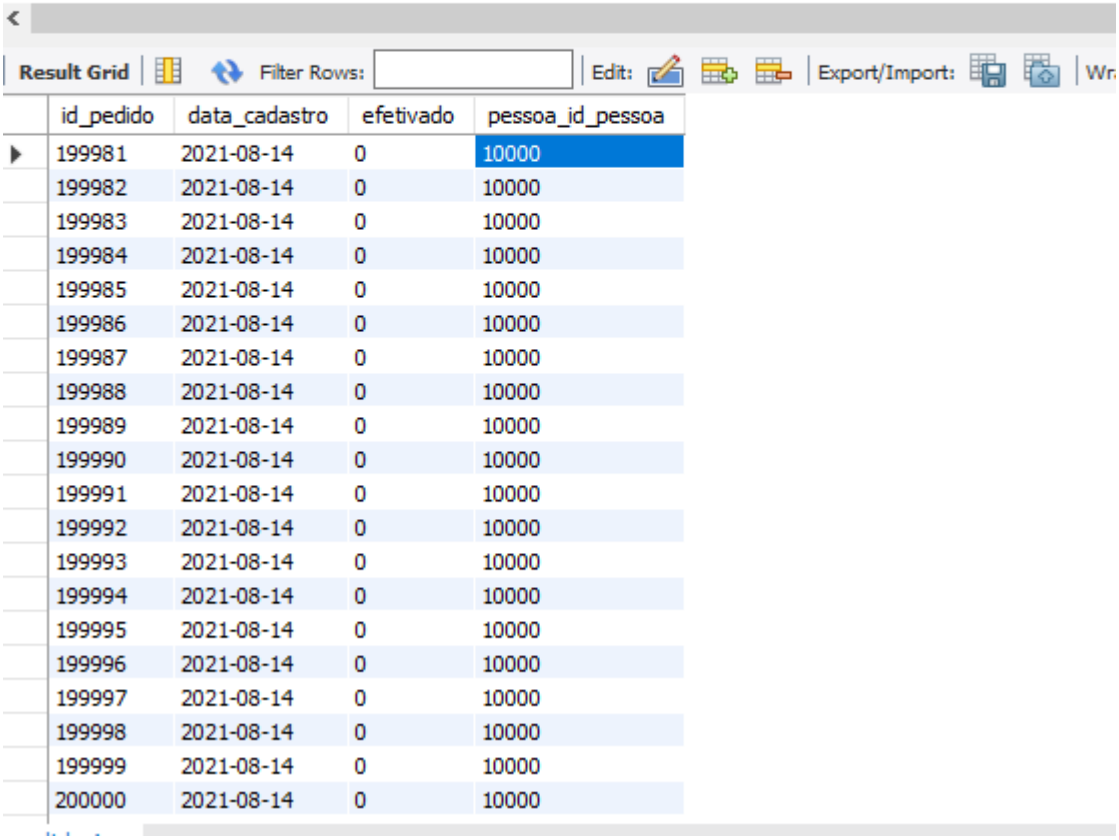
Figura 8 - Tabela de clientes após se tornarem efetivos



Fonte: Elaborado pelo Autor

Figura 9 - Tabela de pedidos após as execuções de trigger e procedures

```
1 •  SELECT * FROM mydb.pedido order by pessoa_id_pessoa desc;
```



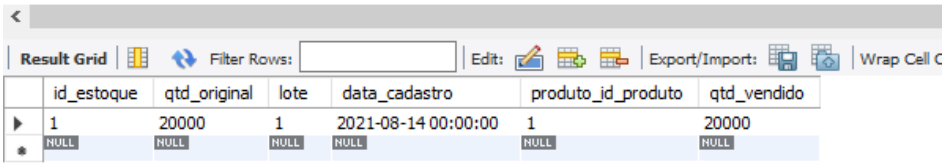
Result Grid | Filter Rows: | Edit: | Export/Import:

id_pedido	data_cadastro	efetivado	pessoa_id_pessoa
199981	2021-08-14	0	10000
199982	2021-08-14	0	10000
199983	2021-08-14	0	10000
199984	2021-08-14	0	10000
199985	2021-08-14	0	10000
199986	2021-08-14	0	10000
199987	2021-08-14	0	10000
199988	2021-08-14	0	10000
199989	2021-08-14	0	10000
199990	2021-08-14	0	10000
199991	2021-08-14	0	10000
199992	2021-08-14	0	10000
199993	2021-08-14	0	10000
199994	2021-08-14	0	10000
199995	2021-08-14	0	10000
199996	2021-08-14	0	10000
199997	2021-08-14	0	10000
199998	2021-08-14	0	10000
199999	2021-08-14	0	10000
200000	2021-08-14	0	10000

Fonte: Elaborado pelo Autor

Figura 10 - Tabela de estoque após as execuções de trigger e procedures

```
1 •  SELECT * FROM mydb.estoque;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

id_estoque	qtd_original	lote	data_cadastro	produto_id_produto	qtd_vendido
1	20000	1	2021-08-14 00:00:00	1	20000
NULL	NULL	NULL	NULL	NULL	NULL

Fonte: Elaborado pelo Autor

Como é possível perceber o resultado foi satisfatório, visto que nessa pequena amostra não houveram erros, e consequentemente não houve perda de dados. Vale ressaltar que em casos de erros, a implementação prévia alguns casos que seriam de fácil rastreamento o que facilita muito para análise de dados futuras, bem como a implementação em um sistema vinculado a esses dados. Com isso é possível dizer que a solução proposta é extremamente eficaz analisando suas vantagens como descrito anteriormente como o resultado final de sua execução.

5 CONCLUSÃO

Em relação à aplicação das regras de negócios diretamente nos objetos do SGBD, com certeza são observadas diversas vantagens. Como a centralização dos dados e de sua manipulação em apenas um local, ter a necessidade de alguém especializado em uma linguagem, ao invés de várias, e a garantia de segurança das informações, já que a execução das *triggers* e *procedures* são automáticas e atômicas. Este último seria uma das maiores vantagens, pois não depende do usuário final para aplicar a alterações, ou então da latência de servidores, que às vezes podem causar falhas.

Vale destacar que o modelo proposto conseguiu reduzir problemas de transações "*lockantes"* e reduzir a complexidade da aplicação já que, não ficaria a cargo do desenvolvedor criar diversos mecanismos e verificações para manter a autenticidade dos dados, ficando tudo centralizado no banco que consegue gerir com as propriedades de atomicidade e consistência (Barros, 2021). Além disso para tarefas complexas, seria necessário aplicar uma otimização no banco a fim de obter melhores resultados em relação a tempo de execução.

6 REFERÊNCIAS

AWS. O que são microsserviços?. Disponível em: <https://aws.amazon.com/pt/microservices/> Acesso em 17 de abril. 2021.

BARROS, P. O que é ACID?. Disponível em:<https://medium.com/opensanca/o-que-%C3%A9-acid-59b11a81e2c6>. Acesso em: 07 de Março 2021.

DATE, Christopher. J. Introdução a sistemas de bancos de dados. Rio de Janeiro: Elsevier, 2003.

DEVMEDIA. Guia Completo de Sql. 2021. Disponível em <https://www.devmedia.com.br/guia/guia-completo-de-sql/38314>. Acesso em: 17 de setembro de 2021.

DRAGONI, Nicola et al. Microservices: yesterday, today, and tomorrow. Cornell University, 2016. Disponível em <https://hal.inria.fr/hal-01631455/document>. Acesso em: 17 de setembro de 2021.

FILHO, Antonio. Desenvolvimento orientado para arquitetura. 2008. Disponível em <https://www.devmedia.com.br/arquitetura-de-software-desenvolvimento-orientado-para-arquitetura/8033> Acessado em 24 de Setembro de 2021.

JÚNIOR, Ary. “Introdução a SQL”. 2008. Disponível em <https://www.devmedia.com.br/artigo-sql-magazine-54-introducao-a-sql/9395> Acessado em 17 de Setembro de 2021.

MONTEIRO, Danielle. Afinal de contas, o que é uma Stored Procedure?. Imaster D. , 2018. Disponível em:< https://imasters.com.br/banco-de-dados/afinal-de-contas-o-que-e-uma-stored-procedure>. Acesso em: 07 de Março 2021.

MELTON, Jim. SQL: The Standard and the Language. 1994. Disponível em: <http://archive.opengroup.org/public/tech/datam/sql.htm>. Acesso em: 10 de setembro de 2021.

MICROSOFT. Estilo de arquitetura de microsserviço. 2021. Disponível em: <https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices> Acessado em 17 de Setembro de 2021

MULOTTO, Cesar; QUEIROZ, Wiluey. Locks, Blocks e Deadlocks – Qual a diferença?. 2018. Disponível em: <https://dbabrasil.net.br/locks-blocks-deadlocks/>. Acesso em: 24 de setembro de 2021

MYSQL. MySQL Customers. 2021. Disponível em: <https://www.mysql.com/customers.> Acesso
em 25 de abril. 2021.

ORACLE, O que é um banco de dados?. 2021. Disponível em :<https://www.oracle.com/br/database/what-is-database/>. Acesso em: 17 de Setembro 2021.

ORACLE. Create Procedure. 2021. Disponível em: <https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_6009.htm>. Acesso em: 23 de abril de 2021.

ORACLE, SQL Developer Supplementary Information for MySQL Migrations. 2021. Disponível em :<https://docs.oracle.com/cd/E39885_01/doc.40/e18461/triggers_proc_mysql.htm#RPTMS144>. Acesso em: 17 de Setembro 2021.

PULGA, Sandra; FRANÇA, Edson; GOYA, Milton. Banco de dados: implementação em SQL, PL/SQL e Oracle 11g. São Paulo. Pearson Education do Brasil, 2013.
SALESFORCE, A Importância de CRM para Pequenas Empresas. Disponível em:<https://www.salesforce.com/br/solutions/small-business-solutions/importancia-crm/>. Acesso em: 07 de Março 2021.

REIS, R. Q.; REIS, A. L.. Bancos de dados orientados a objetos: uma introdução. Revista SQL Magazine, 2008, edição 02, disponível em: . Acesso em: 11/05/2012.

RICHARDSON, Chris. Decomposing Applications for Deployability and Scalability. 2014. Disponível em: < www.infoq.com/articles/microservices-intro > Acessado em 24 de Setembro de 2021.

RODRIGUES, Joel. Stored Procedure Mysql. 2013. Disponível em <https://www.devmedia.com.br/stored-procedures-no-mysql/29030> Acessado em 24 de Setembro de 2021.

TEIXEIRA, José. Introdução ao mysql. 2013. Disponível em <https://www.devmedia.com.br/introducao-ao-mysql/27799> Acessado em 24 de Setembro de 2021.