Documentação da Normalização do Banco de Dados – Trocatine

Sumário

[Informações gerais do Banco 2](#_Toc180142265)

[Tabelas 2](#_Toc180142266)

[Users 3](#_Toc180142267)

[Adresses 4](#_Toc180142268)

[Adresses\_Users 4](#_Toc180142269)

[Phones 4](#_Toc180142270)

[Products 5](#_Toc180142271)

[Tags 5](#_Toc180142272)

[Products\_Tags 5](#_Toc180142273)

[Categories 5](#_Toc180142274)

[Product\_Categories 6](#_Toc180142275)

[Favorites 6](#_Toc180142276)

[Push 6](#_Toc180142277)

[Users\_Push 6](#_Toc180142278)

[Shopping\_Cards 6](#_Toc180142279)

[History\_Shopping\_Cards 7](#_Toc180142280)

[Saved\_Cards 7](#_Toc180142281)

[Orders 7](#_Toc180142282)

[Highlights 7](#_Toc180142283)

[Trocadinhas 8](#_Toc180142284)

[History\_Trocadinhas 8](#_Toc180142285)

[Tabelas de Log 8](#_Toc180142286)

[Log\_Highlights 8](#_Toc180142287)

[Log\_Shopping\_Carts 9](#_Toc180142288)

[Log\_Trocadinhas 9](#_Toc180142289)

[Log\_Users 9](#_Toc180142290)

[Log\_Push 9](#_Toc180142291)

[Log\_Products 10](#_Toc180142292)

[Procedure com transactions 10](#_Toc180142293)

[Create\_User 10](#_Toc180142294)

[Create\_Product 10](#_Toc180142295)

[Create\_Order 11](#_Toc180142296)

[Procedure sem transactions 11](#_Toc180142297)

[Create\_Category 11](#_Toc180142298)

[Create\_Tag 12](#_Toc180142299)

[Functions 12](#_Toc180142300)

[Value\_Product 12](#_Toc180142301)

[Total\_Value\_Order 12](#_Toc180142302)

[Podium\_Trocadinhas 12](#_Toc180142303)

[Triggers com tabelas de Log 13](#_Toc180142304)

[Triggers sem tabelas de Log 13](#_Toc180142305)

[Trigger History Shopping\_Carts 13](#_Toc180142306)

[Trigger History Trocadinhas 13](#_Toc180142307)

[Trigger Update Trocadinhas 14](#_Toc180142308)

[Modificações realizadas entre os bancos 14](#_Toc180142309)

[1° Forma Normal – Não apresentar atributos multivalorados em sua estrutura 14](#_Toc180142310)

[2° e 3° Forma Normal – Todos os atributos não chave deve ser totalmente dependente da chave / um atributo não chave não deve depender de outro atributo não chave 16](#_Toc180142311)

# Informações gerais do Banco

O banco de dados apresentado é contemplado para um sistema relacional gerenciado pelo PostgreSQL, utilizando a plataforma do pgAdmin. Além disso a criação do banco de dados seguirá as seguintes regras de negócio:

* Todas as suas **colunas** e **tabelas** serão nomeadas em inglês.
* Todas as suas **colunas** e **tabelas** serão registradas com letras minúsculas.
* Todas as suas **colunas** serão registradas no singular.
* Todas as suas **tabelas** serão registradas no plural.

Ao total são **25** tabelas, contando tabelas de ligação e tabelas de log, **5** procedures, **3** functions, **9** triggers. Cada uma delas vão ser explicadas a seguir.

## Tabelas

Durante a explicação das tabelas, o formato apresentado será:

* Explicação da serventia da tabela.
* Nome de cada coluna e seus tipos.
* O que cada coluna armazena.

### Users

A tabela de users armazena todos os usuários que são registrados no aplicativo, contêm suas informações gerais e é crucial para ser referenciada a cada dado que diz a respeito de quem está utilizando o programa. Suas colunas seriam:

* id\_user: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* first\_name: VARCHAR(20) NOT NULL CHECK (LENGTH(first\_name) >= 3)

O primeiro nome do user que utiliza o nosso aplicativo, que não pode estar nulo. Além disso, existe uma contagem para saber se tem mais de três letras, considerado o mínimo para preencher a coluna.

* last\_name: VARCHAR(100) NOT NULL CHECK (LENGTH(last\_name) >= 2)

O último nome (sobrenome) do user que utiliza o nosso aplicativo, que não pode estar nulo. Além disso, existe uma contagem para saber se tem mais ou igual a duas letras, considerado o mínimo para preencher a coluna.

* email: VARCHAR(80) NOT NULL UNIQUE

O email do user, ele é utilizado para fazer o login no nosso aplicativo, ele deve ser único e não pode ser nulo exatamente por conta de sua função substancial. Além disso, é verificado se a assinatura do email se encaixa nas seguintes formas:

‘@gmail.com'

'@hotmail.com'

'@mail.com'

'@yahoo.com'

'@outlook.com'

* cpf: VARCHAR(15) NOT NULL UNIQUE

O cpf do user que é utilizado para verificar se a pessoa existe e garantir que ela não se inscreva mais de uma vez no nosso banco. Por este motivo ele precisa estar preenchido e é único.

* birth\_date: DATE NOT NULL

A data de nascimento do user, para saber e ter de forma atualizada a idade do user. Não pode ser nulo, por ser uma informação importante.

* admin: BOOLEAN DEFAULT FALSE

Serve para identificar se o user tem a permissão de administrador no aplicativo, para ter acesso a área restrita. O padrão seria não ter acesso.

* nickname: VARCHAR(20) NOT NULL UNIQUE CHECK (LENGTH(nickname) > 3)

O nickname do usuário no aplicativo, a forma que ele é nomeado no programa. Como ele deve ser obrigatoriamente identificável, não pode ser nulo. Deve ser único para não haver confusões de procura, além disso deve ter mais de 3 letras.

* password: VARCHAR NOT NULL

É a senha que será utilizada pelo user para entrar no programa, ela é criptografada antes da inserção e não pode ser nula por ser substancial. A verificação da quantidade de letras será feita antes de sua inserção no banco por outras camadas do aplicativo.

### Adresses

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_adress: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* street: VARCHAR(80) NOT NULL CHECK (LENGTH(street) >= 5)

Referente a rua do endereço que o user deseja receber seus itens. A coluna não pode ser nula por ser uma informação substancial para encontrar o local indicado e deve ter mais de 5 letras no mínimo considerando a palavra ‘rua’, o espaço em seguida e o nome da rua com, pelo menos, uma letra.

* number: VARCHAR(5) NOT NULL

A respeito do número da residência que o user deseja receber seus itens. A coluna não pode ser nula por ser uma informação ser importante para localizar a construção correta na rua indicada.

* city: VARCHAR(80) NOT NULL CHECK (LENGTH(city) >= 3)

A cidade do endereço que o user deseja receber seus itens. A coluna não pode ser nula por ser uma informação substancial para encontrar o local indicado, já que pode existir outras ruas com o mesmo nome em cidades diferentes. Além disso, deve ter mais de 3 letras considerando que o menor nome de cidade possivel teria esta quantidade de letras.

* state: VARCHAR(80) NOT NULL CHECK (LENGTH(state) >= 4)

O estado do endereço que o user deseja receber seus itens. A coluna não pode ser nula para prevenir ruídos de informação ao procurar a localização indicada. O mínimo de letras foi estipulado considerando que o estado que contêm menos letras.

* neighborhood: VARCHAR(80)

O bairro ao qual pertence a residência indicada, não tem requisitos adicionais por não ser vital na procura da localização, apesar de agregar informações que poderiam deixar ainda mais simples a busca.

* complement: VARCHAR(120)

O complemento seriam indicações a respeito do tipo de construção da residência, como por exemplo, casa, número do apartamento e bloco, sobrado etc. Não tem requisitos adicionais por não ser vital na procura da localização, apesar de agregar informações que poderiam deixar ainda mais simples a busca, apresentando um papel importante apenas quando se trata de prédios.

* cep: VARCHAR(15) NOT NULL CHECK (LENGTH(cep) >= 8)

Referente ao cep do endereço que o user deseja receber seus itens. A coluna não pode ser nula por ser uma informação crucial para encontrar o local indicado, contendo diversas informações relevantes que podem ser consultadas através de seu código. Deve ter mais de 8 letras no mínimo considerando a quantidade de dígitos que um cep carrega, mas suportando dígitos a mais em casos de serem inseridos com pontos e traços.

### Adresses\_Users

A tabela de adresses\_users é uma tabela intermediaria entre as tabelas de users e adresses, ela carrega a ligação de cada usuário com os endereços que ele cadastrou. Suas colunas seriam:

* id\_adress: INTEGER NOT NULL

Armazena o index correspondente ao endereço cadastrado.

* id\_user INTEGER NOT NULL

Armazena o index correspondente ao user que utiliza aquele endereço.

### Phones

A tabela de Phones guarda os números de telefone e a relação do mesmo entre os usuários, já que é possivel um usuário ter mais de um telefone. Estas seriam as suas colunas:

* id\_phone: SERIAL PRIMARY KEY NOT NUL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_user: INTEGER NOT NULL

Armazena o index referente a tabela de **users**, assim facilitando nos casos que as consultas na respectiva tabela são direcionadas pela pessoa e com isso é possivel representar a qual usuário pertence o número de telefone.

* number: VARCHAR NOT NULL UNIQUE CHECK (LENGTH(number) >= 13)

O número do telefone que o user deseja ser contatado, por ser fundamental ele não pode ser deixado em branco. O campo leva como premissa que um número de telefone tem no mínimo 13 dígitos levando em consideração o código de país, código de área (DDD) e número do assinante. Apesar de um usuário poder cadastrar mais de um número, o mesmo número de telefone não pode ser atribuído para mais de um usuário, logo ele é único.

### Products

A tabela de products é o local onde é registrado as informações a respeito de cada produto anunciado no programa. Suas colunas seriam:

* id\_product: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_user: INTEGER NOT NULL

Armazena o index referente a tabela de **users**, assim facilitando nos casos que as consultas na respectiva tabela são direcionadas pela pessoa e com isso é possivel identificar qual usuário anunciou o produto. Não pode ser nulo, pois todo produto é ofertado por algum usuário.

* name: VARCHAR(100) NOT NULL CHECK (LENGTH(name) >= 3)

O nome do produto, ele não pode ser nulo por ser necessário o nome em sistemas como o de procura, e deve ter pelo menos 3 letras.

* description: VARCHAR CHECK (LENGTH(description) >= 3)

A descrição a cerca do artigo, como tamanho, defeitos, características em geral. Deve ter pelo menos 3 letras caso for preenchido.

* value: NUMERIC DEFAULT 0

O valor que o usuário deseja receber por aquele produto em caso de venda. Por conta de o foco do aplicativo ser em trocas, o padrão é ser zerado.

* stock: BIGINT DEFAULT 1

O estoque do produto, comumente terá apenas um produto visando que o objetivo do aplicativo é para produtos usados por isso o seu padrão é ser 1, entretanto esta coluna serve para o caso de haver mais unidades do mesmo produto.

* created\_at: DATE DEFAULT current\_date

A data em que o produto foi registrado no sistema.

* update\_at: DATE DEFAULT current\_date

A data em que o produto foi atualizado pela última vez no sistema.

* flag\_trade: BOOLEAN NOT NULL DEFAULT TRUE

É uma indicação de se o produto é anunciado com a intenção fazer uma troca. Em caso de verdadeiro (true) é uma troca, em caso de falso (false) é uma venda. Por conta de o foco do aplicativo ser em trocas, o padrão é ser verdadeiro (true).

### Tags

A tabela de tags é composta por informações que qualificam itens de variadas formas, dependendo da combinação das informações contidas em cada coluna. Sua formação é pensada para permitir serem utilizados e criados diversos tipos de qualificações que não foram pensados previamente, além de permitir a verificação das outras camadas da construção do aplicativo a respeito das informações contidas, para que o mesmo conjunto não seja registrado mais de uma vez.

* id\_tag: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* type: VARCHAR(80) NOT NULL CHECK (LENGTH(number) >= 3)

O tipo da tag que está sendo registrada. Esses seriam exemplos de valores validos: material, tamanho, cor etc. Ela deve ter no minino 3 letras.

* name: VARCHAR(80) NOT NULL

O nome da tag que está sendo registrada. Esses seriam exemplos de valores validos: madeira, RN, azul etc.

### Products\_Tags

A tabela de products\_tags é uma tabela intermediaria entre as tabelas de products e tags, ela carrega a ligação de cada produto com as tags cadastradas:

* id\_product: INTEGER NOT NULL

Armazena o index correspondente ao produto que é qualificado.

* id\_tag: INTEGER NOT NULL

Armazena o index correspondente a tag cadastrada que o qualifica.

### Categories

A tabela de Categories guarda os nomes das categorias, dos tipos de artigo que os produtos cadastrados podem se encaixar. Suas colunas seriam:

* id\_category: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* name: VARCHAR(80) NOT NULL CHECK (LENGTH(name) >= 3)

O nome das categorias cadastradas, que nicham os produtos. Eles devem ter no mínimo 3 letras.

### Product\_Categories

A tabela de products\_categories é uma tabela intermediaria entre as tabelas de products e categories, ela carrega a ligação de cada produto com as categorias cadastradas:

* id\_product: INTEGER NOT NULL

Armazena o index correspondente ao produto cadastrado que irá ser encaixado em um nicho.

* id\_category: INTEGER NOT NULL

Armazena o index correspondente a categoria cadastrada que demonstra em qual nicho o produto se encaixa.

### Favorites

A tabela de favorites é uma tabela intermediaria entre as tabelas de products e user, ela é utilizada para saber quais são os produtos favoritados de cada usuario, desta forma ela carrega a ligação de cada user com os produtos cadastrados a partir da motivação explicada anteriormente:

* id\_user: INTEGER NOT NULL

Armazena o index correspondente ao user que favoritou o produto.

* id\_product: INTEGER NOT NULL

Armazena o index correspondente ao produto favoritado.

### Push

A tabela de Push guarda as informações a respeito do Armazena o index

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_push: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* title: VARCHAR(100) NOT NULL CHECK (LENGTH(title) >= 3)
* description: VARCHAR NOT NULL
* created\_at: DATE DEFAULT current\_date

### Users\_Push

A tabela de users\_push é uma tabela intermediaria entre as tabelas de users e push, ela carrega a ligação de cada user com os pushs recebidos por ele:

* id\_user: INTEGER NOT NULL

Armazena o index correspondente ao user que recebeu a notificação.

* id\_push: INTEGER NOT NULL

Armazena o index correspondente a notificação que esta cadastrado.

### Shopping\_Cards

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_shopping\_cart\_org: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_shopping\_cart: SERIAL
* id\_user: INTEGER NOT NULL
* id\_product: INTEGER NOT NULL
* quantity: INTEGER NOT NULL

### History\_Shopping\_Cards

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_history\_shopping\_cart: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_shopping\_cart\_old: INTEGER NOT NULL
* id\_user: INTEGER NOT NULL
* id\_product: INTEGER NOT NULL
* quantity: INTEGER NOT NULL
* value: NUMERIC DEFAULT 0
* status: VARCHAR(80) DEFAULT 'Pending'

### Saved\_Cards

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_saved\_card: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_user: INTEGER NOT NULL
* card\_number: VARCHAR(22) NOT NULL CHECK (LENGTH(card\_number) >= 16)
* expiration\_date: DATE NOT NULL
* cvv: VARCHAR(4) NOT NULL CHECK (LENGTH(cvv) >= 3)

### Orders

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_order: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_shopping\_cart\_old: INTEGER NOT NULL
* id\_saved\_card: INTEGER
* payment\_type: VARCHAR(20) NOT NULL CHECK (LENGTH(payment\_type) >= 3)
* total\_value: NUMERIC DEFAULT 0
* accounted\_at: DATE DEFAULT current\_date

### Highlights

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_highlight: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_user: INTEGER NOT NULL
* id\_product: INTEGER NOT NULL
* id\_saved\_card: INTEGER
* payment\_type: VARCHAR(20) NOT NULL CHECK (LENGTH(payment\_type) >= 3)
* value: NUMERIC DEFAULT 0
* accounted\_at: DATE DEFAULT current\_date
* expirantion\_at: DATE NOT NULL

### Trocadinhas

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_trocadinha: SERIAL PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_user: INTEGER NOT NULL
* number\_trocadinha: INTEGER DEFAULT 0
* expiration\_date: DATE NOT NULL
* last\_atualization: DATE DEFAULT current\_date

### History\_Trocadinhas

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* id\_hist\_trocadinha: INTEGER PRIMARY KEY NOT NULL

Armazena o index, esse serve para facilitar as consultas na respectiva tabela e ordena suas linhas de acordo com um identificador sequencial que é entregue para cada uma.

* id\_user: INTEGER NOT NULL
* position: VARCHAR(80)
* number\_trocadinha: INTEGER NOT NULL
* expiration\_date: DATE NOT NULL
* last\_atualization: DATE NOT NULL

## Tabelas de Log

### Log\_Highlights

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* new\_id\_highlight: INTEGER
* update\_at: DATE
* operation: VARCHAR(20)
* user\_changed: VARCHAR(20)
* old\_id\_highlight: INTEGER

### Log\_Shopping\_Carts

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* new\_id\_shopping\_cart: INTEGER
* update\_at: DATE
* operation: VARCHAR(20)
* user\_changed: VARCHAR(20)
* old\_ id\_shopping\_cart: INTEGER

### Log\_Trocadinhas

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* new\_id\_trocadinha: INTEGER
* update\_at: DATE
* operation: VARCHAR(20)
* user\_changed: VARCHAR(20)
* old\_ id\_trocadinha: INTEGER

### Log\_Users

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* new\_id\_user: INTEGER
* update\_at: DATE
* operation: VARCHAR(20)
* user\_changed: VARCHAR(20)
* old\_ id\_user: INTEGER

### Log\_Push

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* new\_id\_push: INTEGER
* update\_at: DATE
* operation: VARCHAR(20)
* user\_changed: VARCHAR(20)
* old\_ id\_push: INTEGER

### Log\_Products

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

* new\_id\_product: INTEGER
* update\_at: DATE
* operation: VARCHAR(20)
* user\_changed: VARCHAR(20)
* old\_ id\_ product: INTEGER

## Procedure com transactions

seriam:

### Create\_User

seriam:

Texto

Descrição gerada automaticamente

### Create\_Product

seriam:

Texto

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

### Create\_Order

ada

Texto

Descrição gerada automaticamente

## Procedure sem transactions

### Create\_Category

Adafds

Texto

Descrição gerada automaticamente

### Create\_Tag

Adafds

Texto

Descrição gerada automaticamente

## Functions

Adafds

### Value\_Product

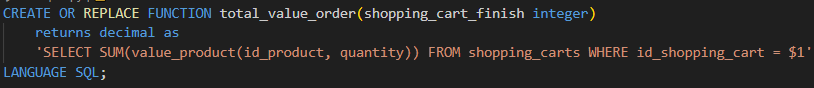
Adafds

Texto

Descrição gerada automaticamente

### Total\_Value\_Order

Adafds



### Podium\_Trocadinhas

Adafds

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

## Triggers com tabelas de Log

### Trigger Highlights Log

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

Texto

Descrição gerada automaticamente

### Trigger Shopping\_Carts Log

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

Texto

Descrição gerada automaticamente

### Trigger Trocadinhas Log

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

Texto

Descrição gerada automaticamente

### Trigger Users Log

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

Texto

Descrição gerada automaticamente

### Trigger Push Log

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

Texto

Descrição gerada automaticamente

### Trigger Products Log

A tabela de adresses armazena o local em que os usuários moram ou desejam receber o seu produto, contêm todas as informações acerca do endereço fornecido. Suas colunas seriam:

Texto

Descrição gerada automaticamente

## Triggers sem tabelas de Log

### Trigger History Shopping\_Carts

Adafds

Texto

Descrição gerada automaticamente

### Trigger History Trocadinhas

Adafds

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

### Trigger Update Trocadinhas

Adafds

Texto

Descrição gerada automaticamente

# Modificações realizadas entre os bancos

A abordagem que vai ser adotada durante a documentação das modificações será conforme cada forma normal, ou seja, vão ser apresentadas conforme a forma normal que motivou cada modificação feita durante o processo.

Sendo assim, prosseguiremos para as ações realizadas durante a normalização.

## 1° Forma Normal – Não apresentar atributos multivalorados em sua estrutura

Tabela User –

Dentro da tabela de usuário apresentava foreign keys que não seguiam essa regra, já que um usuário pode publicar mais de um anúncio, realizar mais de um pedido e ter mais de um endereço, além disso criei uma tabela para telefone já que o usuário pode apresentar mais de um telefone também. Logo a estrutura apresentada assim:

Texto

Descrição gerada automaticamente com confiança baixa

Se tornou isso:

Interface gráfica do usuário, Diagrama, Texto

Descrição gerada automaticamente

Tabela Product (Antigo anuncio) –

Dentro da tabela anúncio era apresentado Foreign Keys que não seguiam as regras, já que dentro de um anúncio é feito diversas avaliações e podem existir mais de uma dúvida, entretanto como essas funções vão ser armazenadas em outro banco, elas foram apenas retiradas. Logo a estrutura apresentada assim:

Uma imagem contendo Texto

Descrição gerada automaticamente

Se tornou isso:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

## 2° e 3° Forma Normal – Todos os atributos não chave deve ser totalmente dependente da chave / um atributo não chave não deve depender de outro atributo não chave

Tabela Tag

A estrutura apresentada pela tabela apresenta que as suas colunas não vão depender totalmente da chave, já que ela tem uma interferência pela Foreign Key de tipo\_produto e as próprias colunas se interferem de acordo com cada anúncio, de acordo com cada combinação. Além disso exclui a possibilidade de um de seus campos não serem utilizados, o que não apresenta coerência já que quando apresentamos um brinquedo, por exemplo, não é possível apresentar um tamanho exato. Logo a estrutura apresentada era assim:

Diagrama

Descrição gerada automaticamente

Diante dessa situação é possível tomar dois caminhos. Poderia ser criada uma tabela para cada coluna e após isso combinar as informações na tabela tag, ou não colocar uma repartição conforme cada tipo armazenando todos na mesma tabela e depois interligando todas as informações diretamente com a tabela anúncio. Foi optado por realizar da segunda forma. Desta forma, se tornou isso:

Diagrama

Descrição gerada automaticamente