Sistema de Gestão de Controle de Salão de Cabelo

DOCUMENTO FINAL DO PROJETO

Empresa: --

Robson de Arruda Silva

Revisões do Documento

Revisões são melhoramentos na estrutura do documento e no seu conteúdo. O objetivo primário desta tabela é a fácil identificação da versão do documento. Toda modificação no documento deve constar nesta tabela.

Data	Versão	Descrição	Autor
18/09/2020	1.0	Documento de Requisitos	Robson
26/10/2020	2.0	Modelo Entidade Relacionamento e Modelo Lógico	Robson
20/11/2020	3.0	Desnormalização e Consultas	Robson
05/12/2020	4.0	Gatilho, Consultas, Procedimento Armazenado e Visão	Robson

1. Visão geral da aplicação e descrição das classes de usuário

A aplicação tem como objetivo gerir o controle de um salão de cabelo, possibilitando que o cliente marque um atendimento desejado e espere a aprovação de um cabeleireiro. Para isso, o sistema disponibilizará os serviços e os horários disponíveis.

Cada serviço incluso no sistema terá uma informação relacionada sobre os produtos utilizados, os quais são obtidos por fornecedores, na qual o usuário poderá ter essa informação detalhada na consulta de um serviço.

O atendimento mostrará ao usuário os dias de trabalhos e os horários disponíveis para tais serviços, seja na consulta de horário ou na marcação de um atendimento.

Além disso, após o serviço ser realizado, o cliente poderá deixar um comentário no sistema a respeito.

1.1 Descrição dos usuários

Os usuários que utilizarão o sistema serão divididos em clientes e cabeleireiros.

1.1.1 Cliente

O usuário cliente poderá fazer consultas de serviços e horários disponíveis, além da marcação de um atendimento, e este, posteriormente, poderá ser desmarcado ou remarcado. Ademais, o usuário cliente também poderá dar seu feedback no sistema sobre determinado serviço.

1.1.2 Cabeleireiro

O usuário cabeleireiro poderá aceitar pedidos de marcação de serviço no sistema, fazer consultas de serviços, atendimentos e horários disponíveis, cadastramento de novos produtos, fornecedores, ferramentas de trabalho e serviços no sistema.

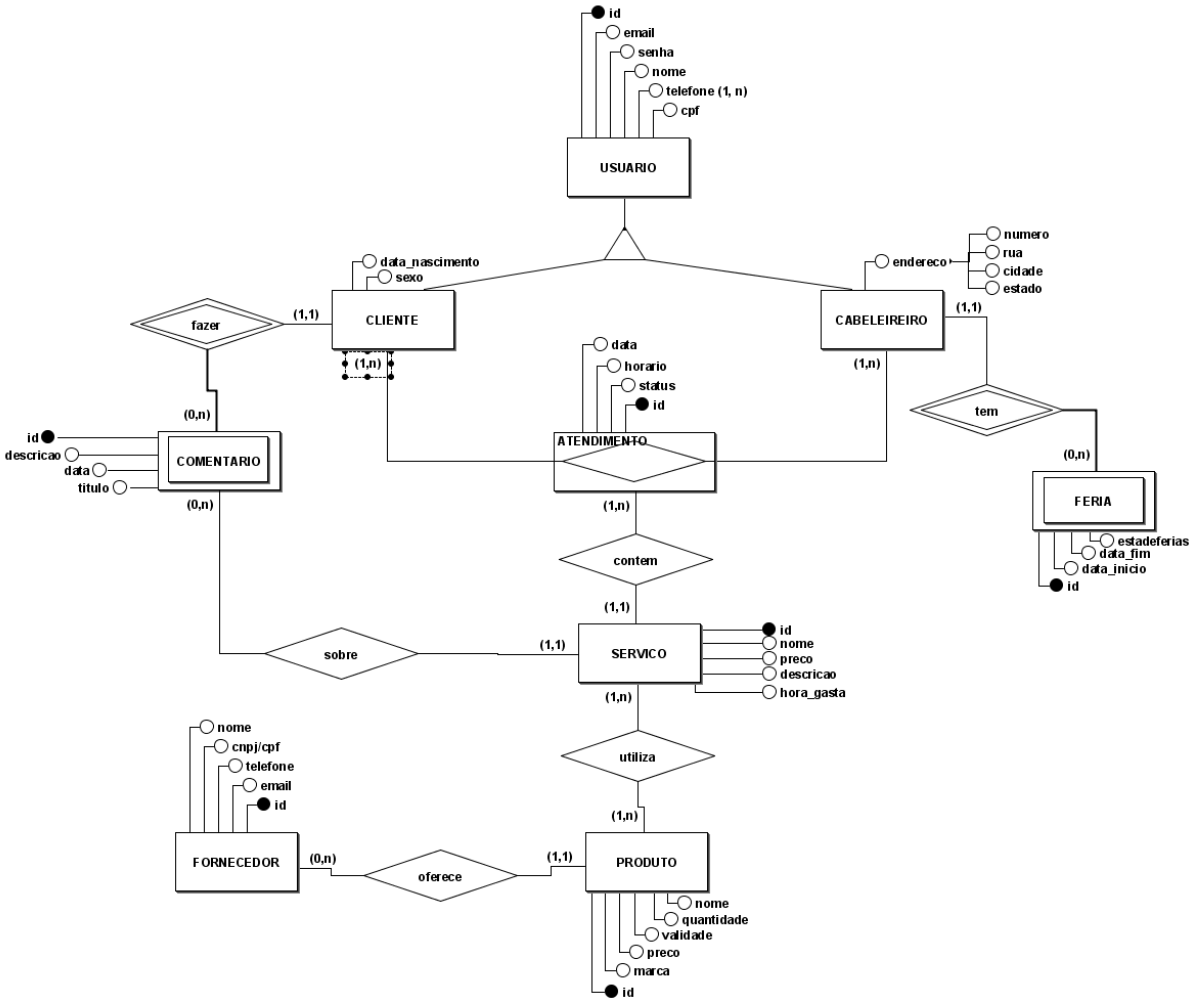
2. Requisitos funcionais

Abaixo estão listados os requisitos funcionais que determinarão as principais funções que serão realizadas pelo sistema em questão de armazenamento de dados.

Número	Descrição	Prioridade
RF1	O s.d.p. manter informações sobre os usuários, contendo os seguintes dados: email, senha, nome, cpf, telefone (podendo ser mais de um), identificador. Os usuários do tipo Cabeleireiro, além das informações de usuário, possuirão: endereço (rua, número, cidade, estado). Os usuários do tipo Cliente, além das informações usuário, possuirão: data de nascimento e sexo. Cada usuário possui apenas um tipo e não existem usuários sem tipo. Cada cabeleireiro estará associado a um cliente e um serviço somente, com isso gerará um atendimento.	(E)

	Cada cabeleireiro poderá ter nenhuma ou várias férias associadas. Ao excluir um cabeleireiro, cada féria associada a ele será excluída. Cada cliente poderá ter nenhum ou vários comentários associados. Cada cliente estará associado a um cabeleireiro e um serviço somente, com isso gerará um atendimento. Ao excluir um cliente, cada comentário associado a ele será excluído.	
RF2	O s.d.p. manter informações sobre os produtos, contendo os seguintes dados: identificador, nome, marca, data de validade, preço, quantidade. Cada produto poderá ter nenhum ou vários fornecedores associados. Cada produto poderá ter um ou vários serviços associados.	(E)
RF3	O s.d.p. manter informações sobre os fornecedores, contendo os seguintes dados: nome, cpf do responsável, telefone, cnpj, email, identificador. Cada fornecedor terá um produto associado.	(E)
RF4	O s.d.p. manter informações sobre os serviços, contendo os seguintes dados: nome, preço, descrição, hora gasta, identificador. O serviço não armazenará dados dos serviços realizados, e sim, dados sobre ele. Um serviço será executado em um atendimento. Cada serviço estará associado a um cabeleireiro e um cliente somente, com isso gerará um atendimento. Cada serviço poderá ter nenhum ou vários comentários associados.	(E)
RF5	O s.d.p. manter informações sobre a féria, contendo os seguintes dados: identificador, data inicial, data final e verificador(se já está de férias ou não). A féria estará associada a um cabeleireiro. Uma féria só irá existir se o cabeleireiro existir.	(E)
RF6	O s.d.p. manter informações sobre os comentários, contendo os seguintes dados: título, data, descrição, identificador. Cada comentário estará associado a um serviço. Cada comentário estará associado a um cliente. Um comentário só irá existir se o cliente existir.	(E)

3. Modelo Conceitual (Modelo Entidade Relacionamento - DER)



4. Modelo Lógico (Modelo Relacional)

usuario (**id**, email, password, nome, cpf)
telefone(**id**, **telefone**)
id referencia usuario(id)
cliente (**id**, data_nascimento, sexo)
id referencia usuario(id)
cabeleireiro(**id** , rua, número, cidade, estado)
id referencia usuario(id)
feria(**id**, data_inicio, data_fim, **id_cabeleireiro**, estadeferias)
id_cabeleireiro referencia cabeleireiro(id)
atendimento(**id**, horario, data, id_servico, id_cliente, id_cabeleireiro, status)
id_servico referencia servico(id)
id _cliente referencia cliente(id)
id_cabeleireiro referencia cabeleireiro(id)
servico(**id**, nome, preco, descricao, hora_gasta)
comentario(**id**, titulo, descricao, data, **id_cliente**, id_servico)
id_servico referencia servico(id)
id_cliente referencia cliente(id)
produto(**id**, nome, marca, preco, validade, quantidade)
produto_servico(**id_produto**, **id_servico**)
id_servico referencia servico(id)
id_produto referencia produto(id)
fornecedor(**id**, nome, email, telefone, cnpj, cpf, id_produto)
id_produto referencia produto(id)

5. Projeto de desnormalização

1) a) Produtos utilizados em um serviço:

Esquema 1:

servico(id, nome)
produto(id, nome, marca, preco, validade, quantidade)
produto_servico(id_produto, id_servico)
id_servico referencia servico(id)
id_produto referencia produto(id)

Tabelas normalizadas para 2FN, onde todos atributos tem dependência funcional total em relação a chave primária.

Esquema 2:
produto_servico(id_servico, id_produto, nome_servico, nome_produto, marca_produto, preco_produto, validade_produto, quantidade_produto)

b) Fornecedor:

Esquema 1:

produto(id, nome, marca)
fornecedor(id, nome, email, telefone, cnpj, cpf, id_produto)
id_produto referencia produto(id)

Tabelas normalizadas para 3FN, onde todos atributos tem dependência funcional total não transitiva em relação a chave primária.

Esquema 2:

fornecedor(id_fornecedor, nome, email, telefone, cnpj, cpf, id_produto, nome_produto, marca_produto)

c) A tabela do esquema 2 da letra **"a"** não está na 2FN porque tem uma dependência funcional parcial de:
id_servico,id_produto → nome_servico(parcial)/ id_servico → nome_servico
id_servico,id_produto → nome_produto(parcial)/ id_produto → nome_produto
id_servico,id_produto → marca_produto(parcial)/ id_produto → marca_produto
id_servico,id_produto → preco_produto(parcial)/ id_produto → preco_produto
id_servico,id_produto → validade_produto(parcial)/ id_produto → validade_produto
id_servico,id_produto → quantidade_produto(parcial)/ id_produto → quantidade_produto

A tabela do esquema 2 da letra **"b"** não está na 3FN porque tem uma dependência

funcional transitiva de:
nome_produto em relação código, pois id_fornecedor ➜ id_produto e id_produto ➜ nome_produto
marca_produto em relação código, pois id_fornecedor ➜ id_produto e numero_produto ➜ marca_produto

6. Consultas (Corrigidas)

- **JUNÇÃO EXTERNA**

  Mostrar todos os clientes que fizeram comentários, bem como os que NÃO fizeram comentários. Os clientes são identificados por seus IDS.

  Junção externa por CONSULTA SQL:

  select cl.id, co.id_cliente from cliente cl left outer join comentario co on cl.id = co.id_cliente;

  Junção externa por AR:

  Cliente _|X| cliente.id = comentario.id_cliente Comentario

- **SELEÇÃO COM LIKE**

  Mostrar apenas os usuários cujos nomes começam com "Ma".

  Seleção com like por CONSULTA SQL:

  select nome from usuario where nome like 'Ma%'

  Seleção com like por AR:

  σ nome = 'Ma%'(usuario)

- **FUNÇÃO DE AGREGAÇÃO COM HAVING**

  Mostrar a quantidade de serviço por título de comentário considerando somente aqueles títulos que tem mais de um serviço.

  Função de agregação com having por CONSULTA SQL:

  select titulo, count(id_servico) from comentario group by titulo having count(id_servico) > 1;

  Função de agregação com having por AR:

  T <- tituloGcount(id_servico) (comentario)

R <- σ id_servico > 1(T)

- **JUNÇÃO INTERNA COM MAIS DE DUAS TABELAS**

  Mostrar o ID dos clientes que fizeram um comentário e que já agendou um atendimento (seja cancelado, realizado ou marcado).

  Junção interna com mais de duas tabelas por CONSULTA SQL:

  select cl.id from cliente cl
  inner join comentario co on cl.id = co.id_cliente
  inner join atendimento at on cl.id = at.id_cliente;

  Junção interna com mais de duas tabelas por AR:

  X <- Πid(cliente |X| cliente.id = comentario.id_cliente comentario)
  Y <- Πid(cliente |X| cliente.id = atendimento.id_cliente atendimento)
  σ X = Y

- **DIVISÃO**

  O email do fornecedor que oferece um produto cujo a marca é "Clear".

  Divisão por CONSULTA SQL:

  select email from fornecedor where id_produto = (select id from produto where marca = 'Clear');

  Divisão por AR:

  Πemail, id_produto(fornecedor |X| fornecedor.id_produto = produto.id produto) / Πid_produto(σ marca = 'Clear'(produto))

6.1. Consultas do PROJETO FINAL

5) i) Junção externa

Mostrar todos os ID dos cabeleireiros que contém alguma féria, bem como os que NÃO contém féria. Os cabeleireiros são identificados por seus IDS.

Junção externa por CONSULTA SQL:

select distinct ca.id, fe.id_usuario from cabeleireiro ca left join feria fe on ca.id = fe.id_usuario;

Junção externa por AR:
Cabeleireiro |X|_ cabeleireiro.id = feria.id_cabeleireiro Feria

ii) Seleção com like
Mostrar apenas os produtos cujos nomes começam com "De".

Seleção com like por CONSULTA SQL:
select nome from produto where nome like 'De%';

Seleção com like por AR:
σ nome = 'De%'(produto)

iii) Função de agregação
Mostrar a soma dos preços dos serviços do salão.

Função de agregação por CONSULTA SQL:
select sum (preco) from servico;

Função de agregação por AR:
G sum(preco) (funcionario)

iv) Função de agregação com having
Mostrar a quantidade (“count”) da quantidade (pode-se dizer, estoque) por marca de produto considerando somente aquelas marcas que tem mais de um produto.

Como exemplo, imagine que tenha dois produtos shampoo com nomes diferentes, por exemplo, “Clear Man” e “Clear Woman”. Esses dois produtos diferentes têm a mesma marca (“Clear”) e também tem a mesma quantidade de itens no estoque (Por exemplo, os dois tem 20 itens no estoque). Essa consulta retorna a marca que eles têm em comum e quantas vezes essa mesma quantidade repete no estoque (20 itens repete duas vezes no estoque). O resultado da consulta é a marca “Clear” e o número 2, porque aparece o número "20 itens" duas vezes no banco.

Função de agregação com having por CONSULTA SQL:
select marca, count(quantidade) from produto group by marca having count(quantidade) > 1;

Função de agregação com having por AR:
T <- marcaGcount(quantidade) (produto)
R <- σ quantidade > 1(T)

v) Junção interna com mais de duas tabelas

Mostrar os cabeleireiros(as) que têm alguma féria marcada e que fará um atendimento.

Junção interna com mais de duas tabelas por CONSULTA:

```
select distinct us.nome from usuario us
inner join feria fe on us.id = fe.id_usuario
inner join atendimento at on us.id = at.id_cabeleireiro;
```

Junção interna com mais de duas tabelas por AR:

X <- Πid(cabeleireiro |X| cabeleireiro.id = feria.id_cabeleireiro feria)

Y <- Πid(cabeleireiro |X| cabeleireiro.id = atendimento.id_cabeleireiro atendimento)

σ X = Y

vi) Operador de conjunto

Mostrar todos os ID dos clientes que fizeram um comentário, já agendou um atendimento (seja cancelado, realizado ou marcado) ou as duas coisas.

Operador de conjunto (UNION) por CONSULTA SQL:

```
(select id_cliente from atendimento) union (select id_cliente from comentario);
```

Operador de conjunto (UNION) por AR:

Πid_cliente(atendimento) U |¨|id_cliente(comentario)

OBSERVAÇÃO: Percebe-se que “atendimento” e “comentário” precisam ser da mesma aridade (o mesmo número de atributos) e os domínios de atributo precisam ser compatíveis, porém, no nosso banco de dados não tinha tabelas que continham essas condições, então fizemos dessas duas tabelas só para exemplo, mesmo não tendo as condições necessárias.

vii) Junção natural (NATURAL JOIN)

Mostrar a junção comparando os atributos iguais das relações feria e atendimento.

Junção natural (NATURAL JOIN) por CONSULTA SQL:

```
select * from feria natural join atendimento;
```

Junção natural (NATURAL JOIN) por AR:

feria |X| atendimento

viii) Consulta aninhada com '='

Mostrar os dados do produto cujo o "id" dele é igual o "id_servico" da tabela "produto_servico" onde o "id_produto" é igual a "2".

Consulta aninhada com '=' por CONSULTA SQL:

select * from produto where id = (select id_servico from produto_servico where id_produto = 2);

Consulta aninhada com '=' por AR:

X <- σ id_produto = 2(produto_servico)

Y <- σ id_servico(produto_servico) = X

σ id = Y(produto)

ix) Consulta aninhada com in ou not in

Mostre todos os ID dos clientes que possuem conta e que não possuem atendimento.

Consulta aninhada com in ou not in por CONSULTA SQL:

select id from cliente where id not in(select id_cliente from atendimento);

Consulta aninhada com in ou not in por AR:

(Πid_cliente(cliente) - Πid_cliente(atendimento))

x) Divisão

A data de nascimento do cliente que tem um "comentário" registrado na data de "2020-08-10".

Divisão por CONSULTA SQL:

select data_nascimento from cliente where id = (select id_cliente from comentario where data = '2020-08-10');

Divisão por CONSULTA por AR:

Πdata_nascimento, id_cliente(cliente |X| cliente.id = comentario.id_cliente comentario) / Πid_cliente(σ data_comentario = '2020-08-10'(comentario))

7. Visão

7.1. Visão atualizada e atualizável:

Uma visão que contenha todos os serviços oferecidos pelo salão, com todas as suas informações, para que o cliente possa consultar. Os atributos da tabela base (servico) desta consulta tem que ser renomeados na visão, para um melhor entendimento do usuário cliente.

Código da implementação da visão atualizada e atualizável:

```
create view serviços as
select id as Código,
        nome as Serviço,
        preco as Preço,
        descricao as Descrição,
        hora_gasta as TempoDuração
from serviço;
```

Códigos para teste da visão atualizável e atualizada:

Inserindo uma tupla nova em serviço:

```
insert into servico values (10, 'Platinado', 30.00, 'Tintura prata no cabelo intercalando entre as mechas', '1:00:00');
```

Visualizando a nova tupla, tanto na view quanto na tabela base:

```
select * from service;
select * from serviços;
```

Modificando um dado da tupla nova na view:

```
update serviços
set preço = '150.00'
where serviço = 'Platinado';
```

Visualizando a nova tupla atualizada, tanto na view quanto na tabela base:

```
select * from serviços;
select * from service;
```

Modificando um dado da tupla nova na tabela base:

```
update servico
set id = '6'
where nome = 'Platinado';
```

Visualizando a nova tupla atualizada, tanto na tabela base quanto na view:

```
select * from service;
select * from serviços;
```

7.2. Visão materializada:

Uma visão que pode ser consultada pelo usuário cabeleireiro, contendo informações de todos os produtos do salão. Essa visão tem o nome do produto, código do produto, marca, preço, fornecedor e telefone do fornecedor.

Código da implementação da materializada:

```
create materialized view produtos as
select pdt.nome as Produto,
        pdt.id as codigo,
        pdt.marca,
        pdt.preco as preco,
        forn.nome as fonecedor,
        forn.telefone
from produto pdt join fornecedor forn on pdt.id = forn.id_produto;
```

Códigos para teste da visão materializada:

Visualizando a visão materializada:

```
select * from produtos;
```

Modificando um preço da tabela base com nome de “produto”:

```
update produto
set preco = '10.50'
where id = '4';
```

Visualizando a visão materializada sem alterações:

```
select * from produtos;
```

Obs.: É preciso dar REFRESH no banco de dados.

Visualizando a visão materializada atualizada com as novas alterações:

```
select * from produtos;
```

8. Procedimento Armazenado

Regra de negócio:

Esse procedimento armazenado deve ser executado diariamente, para poder saber se o funcionário está de férias em relação ao seu dia atual.

Para isso, foi criado o campo “estadeferias” na entidade “Feria”.

- Se a feria estiver com a “data_inicio” menor ou igual a data atual e a “data_final” maior ou igual a data atual, “estadeferias” = V (Verdadeiro);
- Se o item anterior não for correto, “estadeferias = F” (Falso).

Código da implementação do procedimento armazenado:

```
create or replace function estadeferias () returns void as $$
declare f feria%rowtype;
BEGIN
	for f in (select * from feria) loop

		if (f.data_inicio <= CURRENT_DATE and f.data_fim >= CURRENT_DATE) then
			update feria set estadeferias = 'V' where
			id = f.id;
		else
			update feria set estadeferias = 'F' where
			id = f.id;
		end if;

	end loop;
	return;
END;
$$
LANGUAGE 'plpgsql';
```

Chama a função:

```
select estadeferias();
```

Verifica quem está de férias (V = Verdadeiro, está férias e F = Falso, não está férias):

```
select * from feria;
```

9. Gatilho

Regra de negócio:

Gatilho para verificar se o atendimento já está marcado e validar a entrada de dados.

Esse gatilho vai cancelar a operação de inserção ou atualização caso elas firam as seguintes restrições de integridade:

- Se a nova data, horário ou “id_cabeleireiro” já existem significa que o atendimento já foi marcado e a operação deve ser cancelada, mostrando a mensagem “Este atendimento já foi marcado!”;
- Se a nova data ou horário são inválidos ou o “id_cabeleireiro” não existem a operação deve ser cancelada, mostrando a mensagem “Data e horário inválidos ou o cabeleireiro não existe!”.

Código da implementação do gatilho:

```
CREATE OR REPLACE FUNCTION verifica_atendimento() RETURNS TRIGGER AS $$
BEGIN
	IF (NEW.data IN (SELECT atendimento.data FROM atendimento) and
		NEW.horario IN (SELECT atendimento.horario FROM atendimento) and
		NEW.id_cabeleireiro IN (SELECT atendimento.id_cabeleireiro FROM atendimento WHERE horario = NEW.horario and data = NEW.data)) THEN
		RAISE EXCEPTION 'O corte para esse dia e com esse cabeleireiro neste horário não está disponível!';
	END IF;

	IF (NEW.data < CURRENT_DATE OR NEW.data = CURRENT_DATE AND NEW.horario < LOCALTIME OR NEW.id_cabeleireiro NOT IN (SELECT atendimento.id_cabeleireiro FROM atendimento)) THEN
		RAISE EXCEPTION 'Data e horário inválidos ou o cabeleireiro não existe!';
	END IF;

	RETURN NEW;
	END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER atendimento_invalido
BEFORE INSERT OR UPDATE
ON atendimento
FOR EACH ROW
EXECUTE PROCEDURE verifica_atendimento();
```

Adiciona um novo atendimento para testar o gatilho:

insert into atendimento (horario, data, id_servico, id_cliente, id_cabeleireiro, status) values ('14:35:00', '2020-12-27', 2, 7, 3, 'Marcado');

Verifica os atendimentos já existentes:

select * from atendimento;