UNIVERSIDADE DO OESTE DE SANTA CATARINA
CAMPUS DE SÃO MIGUEL DO OESTE
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

LUCAS BARON SPIER
JOÃO GABRIEL DE ABREU
WILLIAN PABLO COLOMBO

SISTEMA PARA PROVEDORES DE INTERNET

São Miguel Do Oeste/SC
2022

LUCAS BARON SPIER
JOÃO GABRIEL DE ABREU
WILLIAN PABLO COLOMBO

SISTEMAS PARA PROVEDORES DE INTERNET

Trabalho Acadêmico dos componentes de Banco de Dados II, Engenharia de Software I e Programação II do curso de Ciências da Computação da Universidade do Oeste de Santa Catarina no campus de São Miguel do Oeste.
Professores: Roberson Junior Fernandes Alves, Franciele Carla Petry e Otilia Donato Barbosa

São Miguel Do Oeste/SC
2022

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Esse trabalho apresenta o desenvolvimento de um sistema de gerenciamento para uma provedora de internet. A elaboração de um software pode ser dividido em algumas partes: o desenvolvimento do código em linguagem de programação, a estruturação de um banco de dados e ao longo disso, utilizam-se conhecimentos de engenharia de software como os métodos ágeis e diagramas, para que o projeto se mantenha organizado e explicativo aos não participantes diretos.

Nesse projeto, foi utilizado tecnologias e conhecimentos adquiridos de acordo com a área da ciência da computação. Desenvolvido a partir da orientação a objetos, nesse caso com a linguagem Java, por meio do Eclipse, e também, sobre um banco de dados relacional, por meio do PostgreSQL.

Serão apresentados neste artigo, os métodos, *scripts,* diagramas e a descrição de cada fase do projeto, explicando a realização de cada tarefa. O banco de dados e os relatórios SQL são os mesmos utilizados na primeira fase desse projeto realizado na disciplina de Banco de Dados I, denominado *Inside Provider,* e a partir desse *database*, foram realizadas atualizações no modelo lógico relacional e adicionados os requisitos técnicos solicitados no trabalho.

2 DESENVOLVIMENTO

O devido trabalho, foi planejado com base em três matérias do curso de Ciência da Computação, sendo elas Banco de Dados II, Engenharia de Software I e Programação II. Por este motivo, o desenvolvimento teórico foi elaborado e dividido da mesma forma.

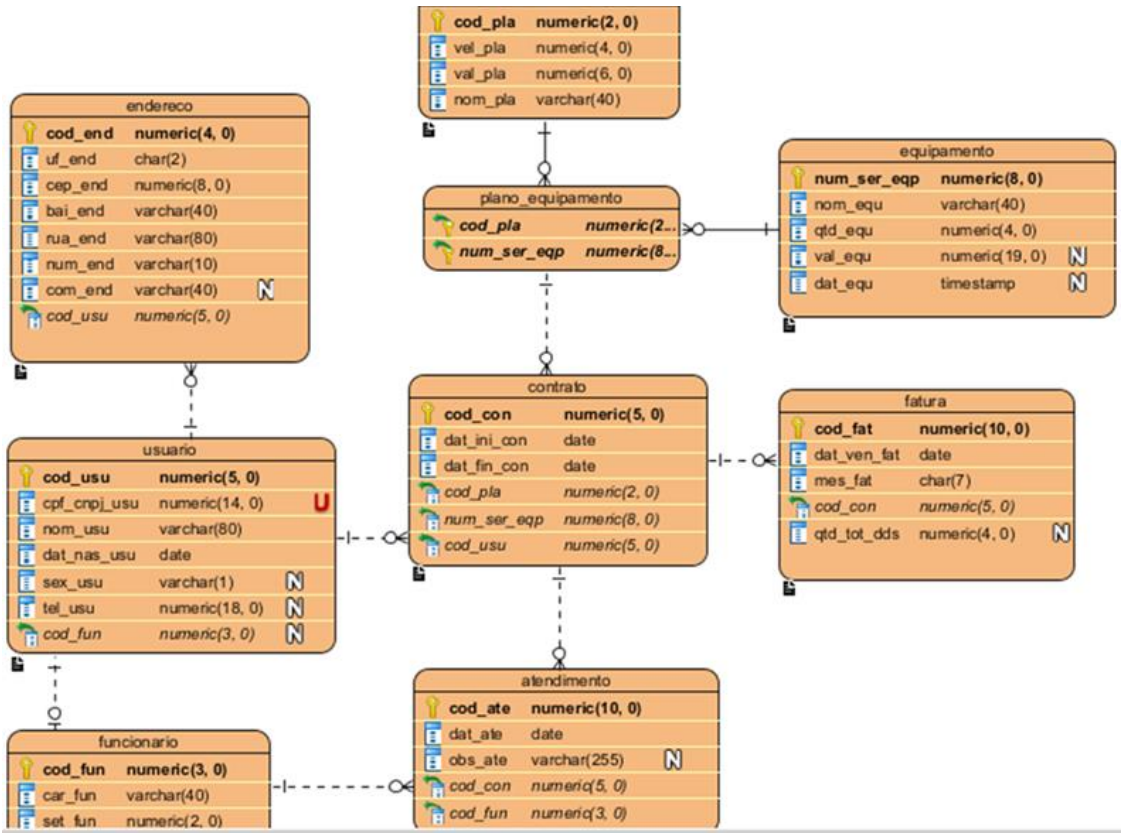
2.1 BANCO DE DADOS II

2.1.1 Modelo Relacional

O primeiro passo para a continuar a elaboração deste projeto, foi atualizar o modelo lógico relacional. Este modelo é importante para os desenvolvedores, pois permite uma fácil identificação das tabelas e atributos, permitindo visualizar os

relacionamentos de cada entidade. Desde sua primeira versão, ele sofreu algumas atualizações, tanto em questão de atributos, quanto na criação e exclusão de tabelas.

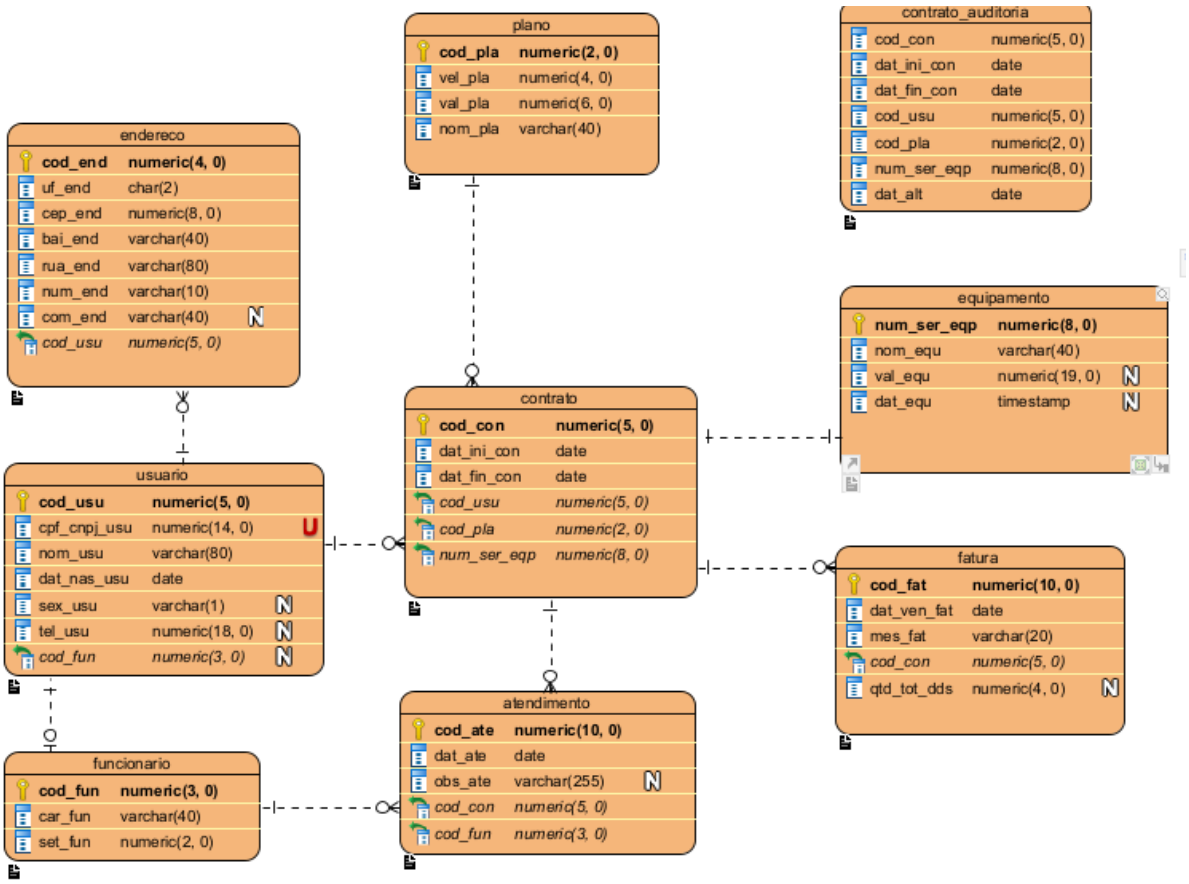
Figura 1: Primeira versão modelo relacional



Fonte: Os autores (2022).

Foi realizada uma análise sobre a tabela principal e seus relacionamentos, notando-se algumas ambiguidades. Dito isso, decidiu-se atualizar as relações entre as tabelas contrato, equipamento e plano, ocasionando a exclusão da tabela “plano_equipamento”, como pode ser visto na Figura 1 e Figura 2.

Figura 2: Modelo relacional atualizado



Fonte: Os autores (2022).

Outra grande mudança, foi a adição da tabela “contrato_auditoria”. Esta entidade foi inserida no modelo, após o desenvolvimento de um mecanismo de auditoria sobre a tabela contrato, na qual será melhor abordado na seção de *triggers*.

2.1.2 Atualização dos scripts

Após as mudanças no modelo relacional, seguiram as atualizações para os *scripts.* Algumas alterações foram realizadas, para que a criação do banco de dados se mantivesse conciso ao DER.

2.1.3 Índices

Para a criação dos índices, como o banco de dados é teórico e não está trabalhando com uma grande massa de registros, não foi preciso uma análise elaborada para a implementação destas. Os índices foram adicionados em todas as chaves estrangeiras, e nas colunas que serão mais utilizadas na prática, como pode ser observado no Apêndice A.

2.1.3 Controle de acesso

Levando em consideração o âmbito interno dos provedores, foi estipulado algumas áreas dentro do ambiente. Dentre estes setores estão o comercial, suporte, financeiro e o administrador, como pode ser visto na Figura 3.

Figura 3: Esquema do controle de acesso

Objetos de negocio	Comercial	Suporte	Financeiro	Administrador	
Endereço	C-A-V	A-V	V	T	
Usuario	C-A-V	A-V	V	T	**Legenda**
Funcionario	-	-	V	T	C - Cadastrar
Atendimento	V	V	V	T	E - Excluir
Contrato	C-A-V	V	V	T	A - Atualizar
Fatura	V	V	C-A-V	T	V - Visualizar
Plano	V	V	V	T	T - Todos

Fonte: Os autores (2022).

Com base na divisão de setores dentro do provedor, foi estimado os privilégios de acesso para cada tabela do banco de dados. Os privilégios foram divididos em cadastrar, excluir, atualizar, visualizar e todos.

Após a elaboração teórica do controle de acesso, foi posto em prática na base de dados. O *script* foi dividido em algumas etapas, começando pela criação das *roles* com base nos setores, e atribuições das permissões para estas. Por último, foi criado um usuário para cada *role*, atribuindo login e senha. O script pode ser visualizado no Apêndice B.

2.1.4 Triggers

Os gatilhos, ou do inglês *triggers*, são mecanismos ativados automaticamente antes ou depois de uma ação pré estabelecida. As *triggers* podem ser utilizadas para estabelecer algumas regras de negócio, auditoria, controle das regras de integridade, etc. No PostgreSQL, elas vêm acompanhadas de uma *procedure* ou uma *function.*

O primeiro gatilho implementado, foi para verificar algumas condições perante à fatura. A *trigger* criada, é acionada antes da tentativa de se inserir ou atualizar a tabela fatura, na qual chama uma função intitulada “verificaFatura”. Essa *function*, testa primeiramente se a fatura em questão já existe, por meio do código, e depois o consumo de dados, na qual não pode ser nulo nem menor que um. O *script* do gatilho pode ser visualizado no Apêndice C.

O segundo gatilho implementado, faz a auditoria da tabela contrato, na qual julga-se ser o pilar principal do banco de dados. A auditoria em questão, armazena alterações feitas na entidade contrato, e para isso, foi preciso adicionar uma nova tabela no modelo e posteriormente aplicada ao próprio banco. Esta tabela não possui relacionamento com outras, sendo utilizada apenas pela chamada da função. O gatilho é acionado depois da inserção ou exclusão de alguma tupla na tabela contrato, na qual chama uma procedure que adiciona esta mesma tupla na entidade “auditoria_contrato” com a data da alteração. O *script* do gatilho pode ser visualizado no Apêndice D.

2.1.5 Functions

Com base nas regras de negócios, foi implementado duas *functions* para o banco de dados. A primeira função, lista todos os usuários que são apenas clientes, excluindo usuários que são funcionários. Foi elaborado de uma maneira simples, na qual utiliza-se de uma estrutura de condição *IF* para verificar se a coluna “cod_fun” na tabela usuário é nulo. A função pode ser vista no Apêndice E.

A segunda *function*, implementa uma consulta sobre os atendimentos em relação ao usuário desejado. Foi desenvolvido uma consulta *select* com auxílio de *joins*, para conseguir unir as tabelas usuario, contrato e atendimento, como pode ser visto na Figura 4.

Figura 4: Unindo as tabelas usuario, contrato e atendimento.

```
select a.cod_ate, a.dat_ate, a.obs_ate from usuario u
    inner join contrato c on u.cod_usu = c.cod_usu
    inner join atendimento a on c.cod_con = a.cod_con
    order by u.cod_usu;
```

Fonte: Os autores (2022).

Após concluir a base da consulta, foi necessário desenvolver a *function* em questão. Para que possa ser escolhido a qual usuário deseja-se pesquisar os atendimentos, foi necessário fazer um ajuste na consulta apresentada anteriormente, na qual adicionou-se um *where* comparando com o parâmetro da função. O *script* detalhado pode ser visto no Apêndice F.

2.1.6 Consultas

Em relação às consultas solicitadas, houveram poucas alterações desde o início do projeto. Os *selects*, sofreram apenas algumas mudanças na estrutura dos *joins,* para que continuassem funcionando como necessário. Essas mudanças foram impactadas pela exclusão da tabela “plano_equipamento”.

Diferente da sua primeira versão, agora as consultas foram transformadas em views, na qual podem ser vistas no Apêndice G.

2.1.7 Backup e Restore

No que diz respeito a backup e *restore,* foram criados *scripts* para realização semi-automática de ambos. Primeiramente foi desenvolvido o plano para realizar backup, que basicamente seria criar uma pasta com o nome “Backup_InsideProvider” em um diretório específico, e depois acessá-lo. Dentro da pasta destino, será criado um documento de texto e um arquivo “.bak”, na qual um fará os registros das ações tomadas pelo backup, e o outro, respectivamente, armazenará os dados para serem restaurados no futuro. O processo de gravação dos dados, é feito utilizando uma função do próprio PostgreSQL, chamada “pg_dump”. O *script* de criação pode ser visto no Apêndice H.

Já o *script* para restauração, fará a leitura do backup pronto. Antes de começar o processo em si, é necessário deletar o banco de dados para que possa ser restaurado sem nenhum problema. O próximo passo, é acessar a mesma pasta destino que foi salvo os dados, na qual fará a leitura e execução do arquivo “.bak”. Após os processos realizados, todas as tabelas, colunas, relacionamentos e registros que estavam no backup, estarão de volta ao banco de dados. O *script* pode ser visto no Apêndice I.

Para uma melhor experiência, foi desenvolvido arquivos “.bat”, tanto para o processo de backup, quanto *restore*. Utilizou-se de um *template* simples com um menu interativo para torná-lo o mais intuitivo possível.

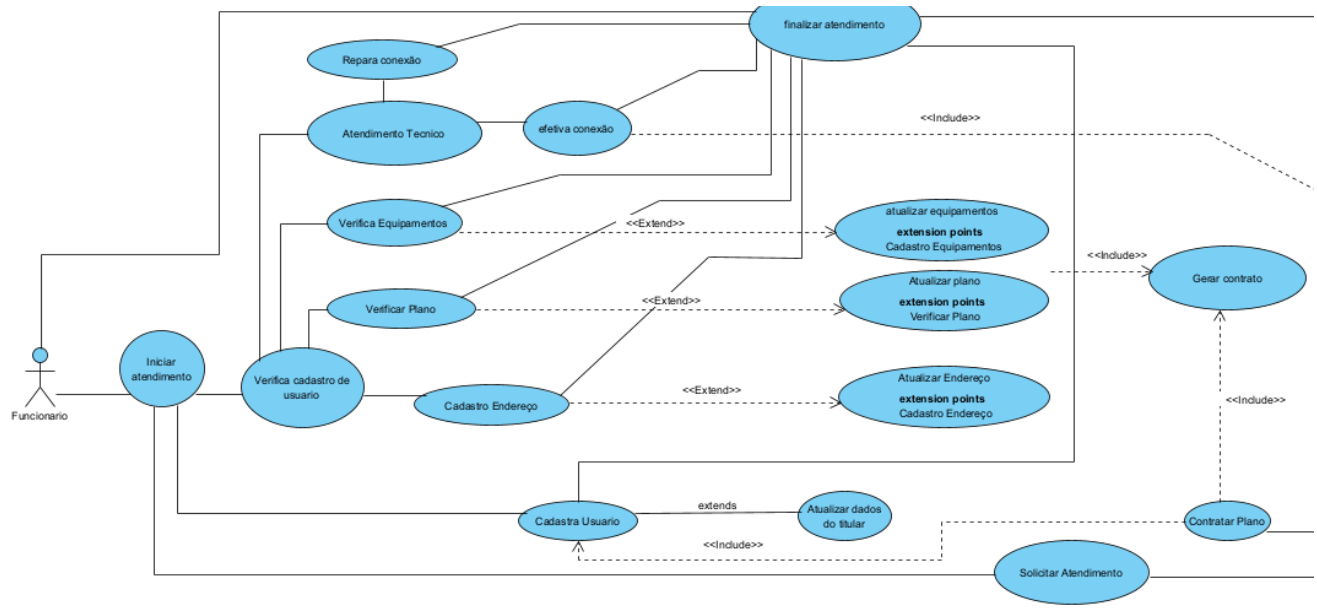
2.2 ENGENHARIA DE SOFTWARE I

Para cada fase do projeto, foram realizados alguns diagramas que auxiliam o entendimento desde os estágios iniciais dos requisitos do sistema até o desenvolvimento do *software*. Estes diagramas são representações que demonstram visualmente o funcionamento do *software.*

2.2.1 Diagrama caso de uso

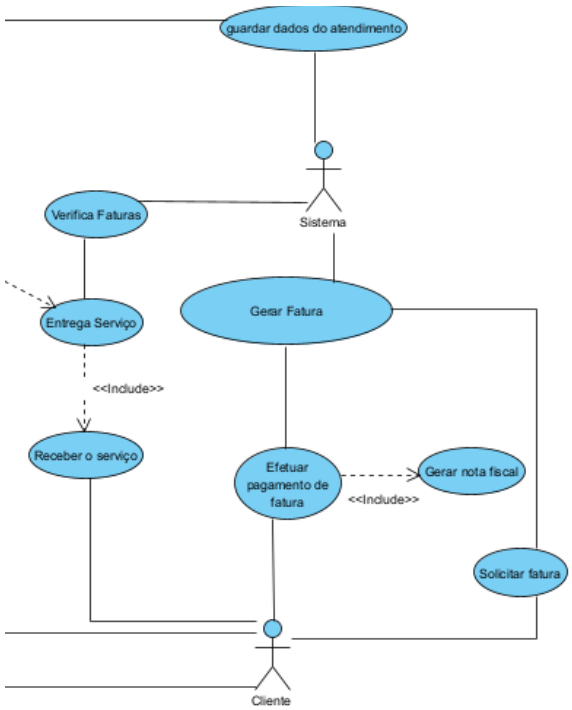
Representa as ações possíveis de cada “ator”, nesse caso de uso (Figura 5 e Figura 6): Funcionario, Cliente e Sistema são os atores. As setas que possuem *extend*s, significam que existe a opção de realizar uma ação, e a seta *Include*, indica obrigatoriedade de ação. As principais ações do ator funcionário são iniciar atendimento, verificar e realizar as ações necessárias a cada demanda e finalizar atendimento. O cliente solicita atendimento, solicita uma fatura pelo sistema, faz o pagamento da fatura, e recebe o serviço. O sistema nesse caso de uso tem a função de guardar os dados do atendimento, verificar as faturas dos clientes, gerar nota fiscal e gerar fatura.

Figura 5: Diagrama caso de uso parte 01.



Fonte: Os autores(2022).

Figura 6: Diagrama caso de uso parte 02.



Fonte: Os autores (2022).

2.2.2 Fluxo de caso de uso

É um fluxo de eventos, descrevendo cada fase do caso de uso, é composto por um fluxo principal, sem nenhum desvio tomado, e por fluxos alternativos, se necessário. Nesse caso (Figura 7), foram feitos dois fluxos principais, de atendimento, onde o cliente se comunica com o funcionário para solicitar um atendimento, e o de Faturamento, onde o cliente solicita a fatura direto pelo sistema.

Figura 7: Exemplo de Fluxos de caso de uso

Fluxo de Caso de uso: atendimento

1. Cliente solicita atendimento
2. Funcionário inicia atendimento ao cliente
3. Funcionário verifica o cadastro do cliente
4. Cliente solicita algum serviço disponível, que pode ser: atendimento técnico, mudança de plano, equipamento ou de endereço
5. Funcionário abrirá o atendimento conforme o que foi solicitado pelo cliente (efetivar ou reparar conexão, atualização de plano, atualização de endereço ou atualização de equipamento)
6. Funcionário termina o atendimento conforme validado pelo cliente
7. Sistema Guarda os dados do atendimento.

Fluxo de Caso de uso: Faturamento

1. Cliente solicita fatura
2. Sistema gera fatura
3. Cliente recebe fatura
4. Cliente Paga a fatura
5. Sistema gera nota fiscal
6. Sistema verifica fatura e entrega serviço para o cliente correspondente ao mês pago

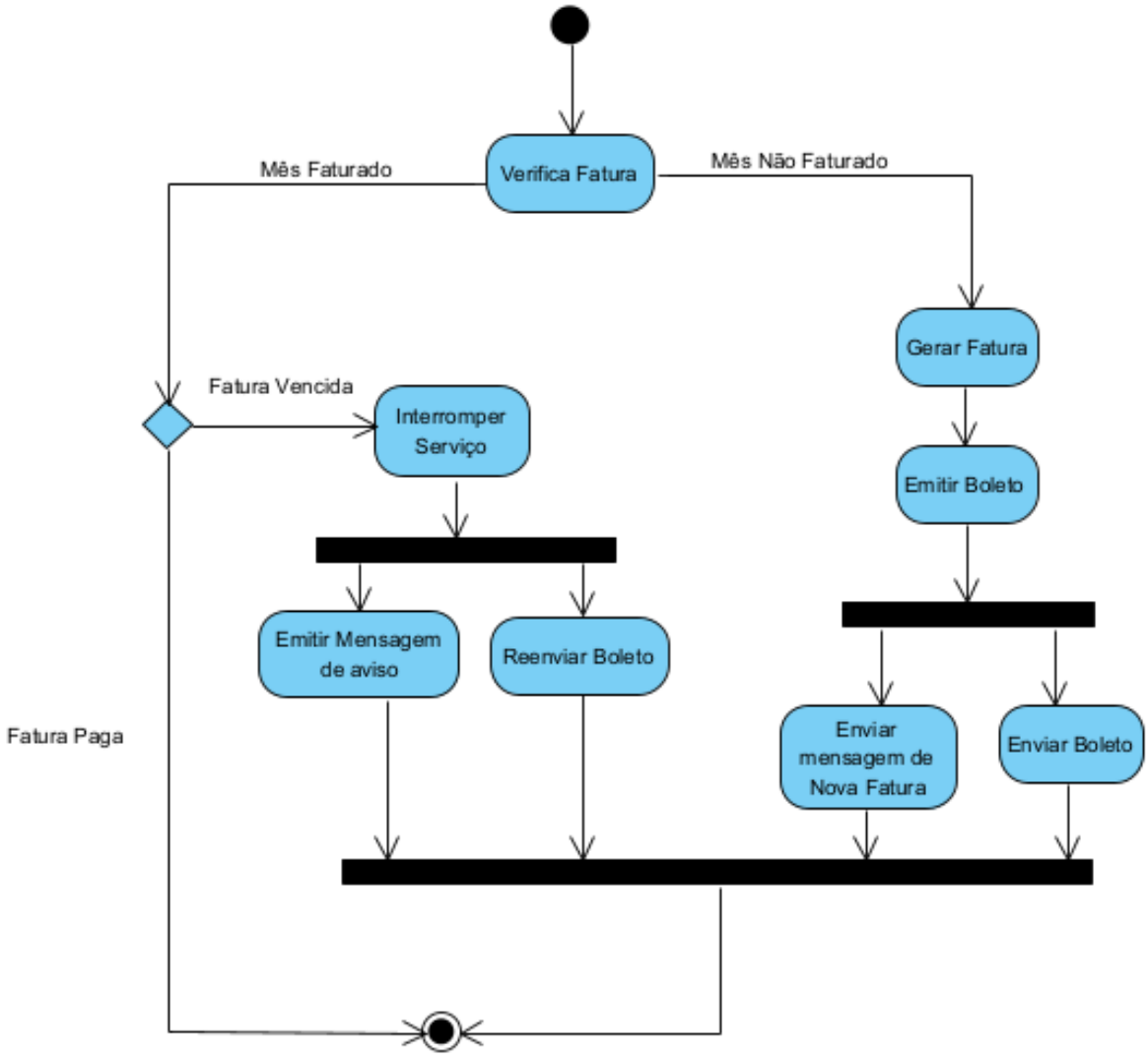
Fonte: Os autores(2022).

2.2.3 Diagrama de atividade

Representa o fluxo de tarefas executadas por algum ator. O sistema no caso de faturamento (Figura 8) primeiramente verifica a fatura, se o mês já foi faturado ele segue por um caminho, e faz outra verificação onde analisa se a fatura foi paga ou não. Em caso de fatura vencida, o sistema interrompe o serviço do cliente realizando duas ações: emitindo uma mensagem de aviso e enviando mais uma vez o boleto

para o cliente. Em caso da fatura já ter sido paga, a tarefa é finalizada. No caso do mês ainda não faturado, o sistema irá gerar a fatura e emitir o boleto para o cliente, a partir disso geram 2 ações ao mesmo tempo: enviar um aviso de nova fatura ao cliente e enviar o boleto para o cliente, assim, finalizando a tarefa.

Figura 8: Diagrama de Atividades.



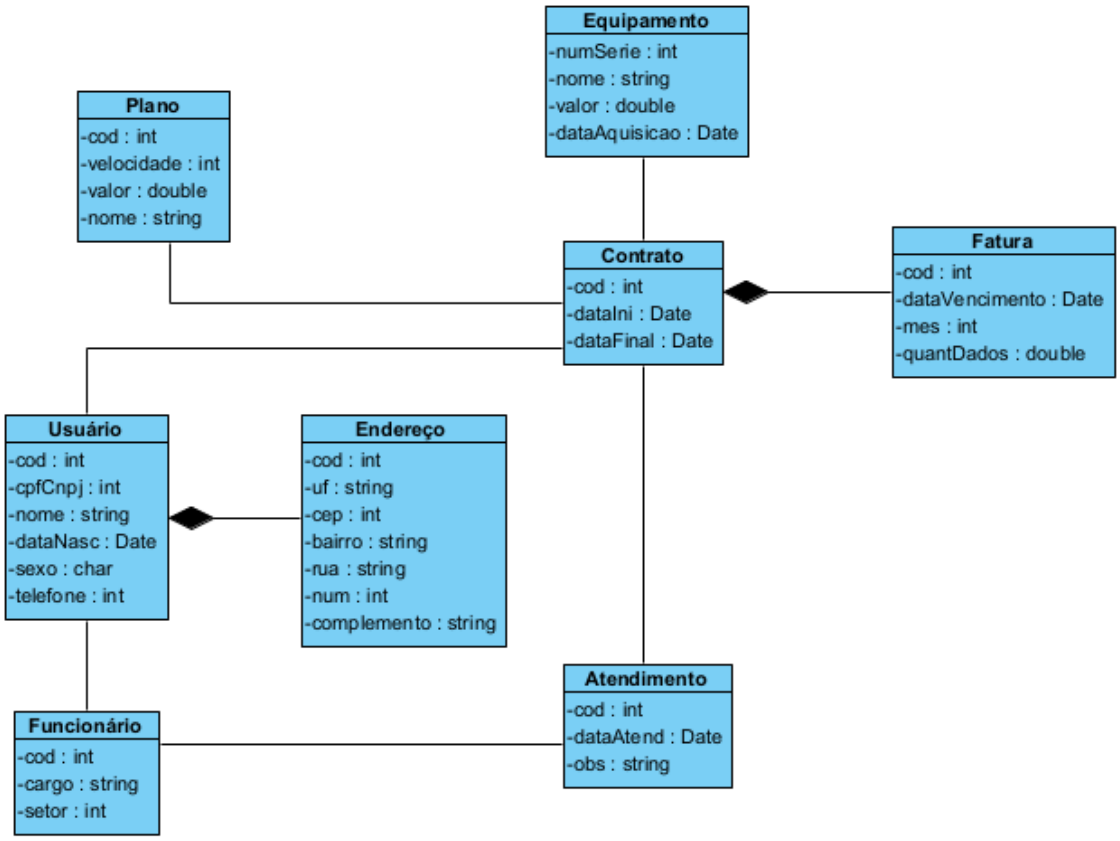
Fonte: Os autores(2022).

2.2.4 Diagrama de classes

O diagrama de classes é o responsável por representar todas as classes e os relacionamentos desses. Os relacionamentos nesse projeto são em sua maioria por associação, apenas dois são por composição, que é quando uma classe

obrigatoriamente precisa de outra para existir, que é o de endereço-usuário e o fatura-contrato. Possível verificar na figura 9.

Figura 9: Diagrama de classes.

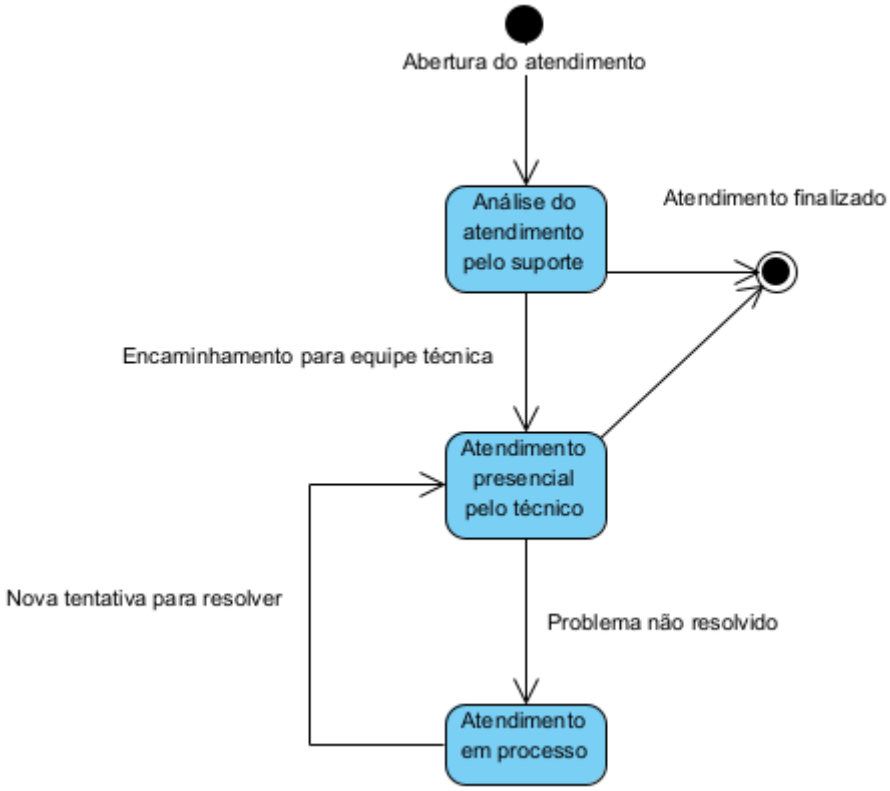


Fonte: Os autores(2022).

2.2.5 Diagrama de estados

Esse diagrama representa os estados em que um objeto pode estar e os gatilhos que promovem essa transição de um estado para outro. Neste diagrama (Figura 10) é realizada a abertura do atendimento, após isso, o funcionário (atendente) irá analisar esse atendimento e caso for necessário, o atendimento irá para um atendimento presencial domiciliar, e, se ainda assim não for resolvido, esse atendimento ficará em aberto, até que sejam realizadas outras tentativas de resolução, ou seja, a classe atendimento possui três estados, quando ele é aberto, quando está em processo, e quando está finalizado.

Figura 10: Diagrama de Estados.

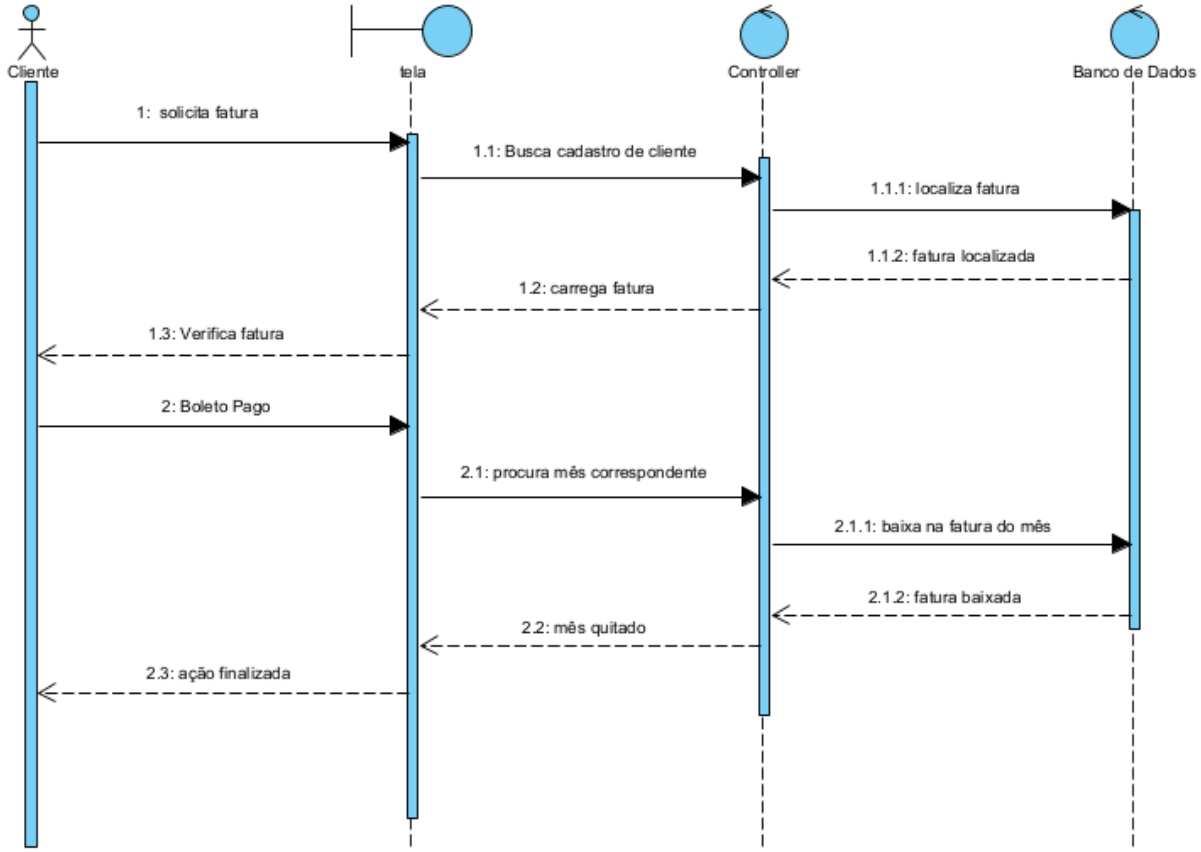


Fonte: Os autores(2022).

2.2.6 Diagrama de sequência

Representa uma troca de mensagens entre os objetos, a realização de uma tarefa distribuída por tempo e por quem realiza cada ação. No diagrama da Figura 11, é demonstrada a tarefa de solicitação e pagamento de uma fatura, passando por todos os momentos cruciais e pelos objetos que fazem parte dessa tarefa.

Figura 11: Diagrama de sequência.



Fonte: Os autores (2022).

2.3 PROGRAMAÇÃO II

2.3.1 Java application

Na construção da classe Application.java, daremos início ao nosso programa. Utilizando a notação @EntityScan, foi mapeado as classes *model,* que serão utilizadas para rodar o projeto Maven, a partir desse arquivo. A classe Java Application, pode ser vista na Figura 12.

Figura 12: Classe Application.

```
* @author InsideProvider
* @version 1.00
* @since Release da aplicação
*/

@EntityScan(basePackages = "br.edu.unoesc.springboot.insideprovider.model")
@SpringBootApplication
public class Application {

    /**  Metodo Main da aplicacao
     * @param args - inicia aplicacao
     */
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Fonte: Os autores (2022).

2.3.2 Classes model

Já nas classes *model*, tem-se os objetos que foram utilizados, como por exemplo a Classe Endereco. Nelas, localizam-se os atributos, construtores*, getters and setters* e a implementação *Serializable,* que possibilita fazer o uso do JpaRepository. Um exemplo de classe *model*, pode ser visto na Figura 13.

Figura 13: Classe Usuario.

```
/**Classe para objetos do tipo Usuario, onde serão contidos, valores e métodos para o mesmo.
 * @author InsideProvider
 * @version 1.00
 * @since Release da aplicação
 */


@Entity
@Table(name="usuario")
public class Usuario implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    @Column(name="cod_usu")
    private Long codUsu;

    @Column(name="cpf_cnpj_usu")
    private Double cpfCnpjUsuario;

    @Column(name="nom_usu")
    private String nomUsu;

    @Column(name="dat_nas_usu")
    private Date nascimentoUsuario;
```

Fonte: Os autores (2022).

2.3.3 Interface Repository

Nas interfaces *repository*, foi utilizado a notação @Repository e estendido para JpaRepository. Também pode ser adicionado comandos SQL, com a notação @Query nesta interface, na qual servirá de base para as interfaces *service*. Um exemplo de interface *repositor*, pode ser visto na Figura 14.

Figura 14: Interface UsuarioRepository.

```
/**Interface para Repository de Usuario.
* @author InsideProvider
* @version 1.00
* @since Release da aplicação
*/

@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

}
```

Fonte: Os autores (2022).

2.3.4 Interfaces service

Também foram implementadas interfaces *service*, para cada classe *model.* Irão trazer as notações de métodos utilizados pela camada *controller*, fazendo com que se tenha um CRUD básico para cada classe. Exemplo de interface *service*, na Figura 15.

Figura 15: Interface UsuarioService.

```
/**Interface para Service de Usuario.
* @author InsideProvider
* @version 1.00
* @since Release da aplicação
*/

public interface UsuarioService {

    List<Usuario> findAllUsuario();
    Optional<Usuario> findAllById(Long codUsu);
    Usuario saveUsuario(Usuario usuario);
    Usuario updateUsuario(Usuario usuario);
    void deleteUsuario(Long codUsu);
}
```

Fonte: Os autores (2022).

2.3.5 Classes service implementation

Para poder utilizar da inteface *service,* foram criadas as classes *service implementation*. Na qual, implementam o *service*, e serão sobrescritos os métodos, de modo que se tenha todos as funções necessárias para listar, filtrar, salvar e deletar

os objetos de nossas classes model. Exemplo de classe *service implementation* presente na Figura 16.

Figura 16: Classe UsuarioServiceImpl.

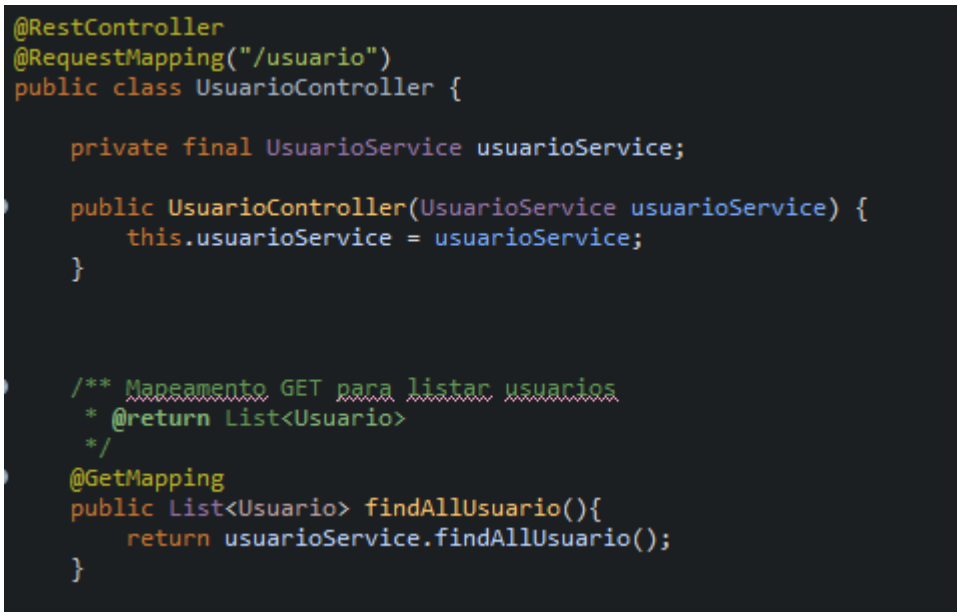
```java
/**Classe para Implementar o Service de Usuario.
 * @author InsideProvider
 * @version 1.00
 * @since Release da aplicação
 */

@Service
public class UsuarioServiceImpl implements UsuarioService {

    private final UsuarioRepository usuarioRepository;

    public UsuarioServiceImpl(UsuarioRepository usuarioRepository) {
        this.usuarioRepository = usuarioRepository;
    }


    /** Metodo para listar usuarios
     * @return List<Usuario>
     */
    @Override
    public List<Usuario> findAllUsuario() {
        // TODO Auto-generated method stub
        return usuarioRepository.findAll();
    }
```

Fonte: Os autores (2022).

2.3.6 Classes controller

Nas classes *controller*, tem-se uma parte muito importante do programa. Para isso, define-se um objeto *service* da classe desejada, e é construído todos os métodos GET, POST, PUT, DELETE das classes que serão acessados via API. Isto é feito através do Postman, ou implementadas na Web, em conjunto com funções JavaScript. Um exemplo de classe *controller,* pode ser visto na Figura 17.

Figura 17: Classe UsuarioController.



```java
@RestController
@RequestMapping("/usuario")
public class UsuarioController {

    private final UsuarioService usuarioService;

    public UsuarioController(UsuarioService usuarioService) {
        this.usuarioService = usuarioService;
    }



    /** Mapeamento GET para listar usuarios
     * @return List<Usuario>
     */
    @GetMapping
    public List<Usuario> findAllUsuario(){
        return usuarioService.findAllUsuario();
    }
```

Fonte: Os autores (2022).

3 CONCLUSÃO

Conclui-se, portanto, que durante o desenvolvimento de um software é preciso ter atenção em todas as etapas para garantirmos a melhor entrega possível do sistema sem problemas estruturais e com uma boa documentação que facilitará futuras pessoas que venham a trabalhar no projeto.

Deve-se dar importância na observação do problema a ser solucionado, no levantamento de requisitos, na elaboração dos modelos, diagramas, na construção dos scripts e, principalmente, na organização e no armazenamento destas informações.

APÊNDICE A - *Script* da criação dos índices

```
-- Criação dos índices nas chaves estrangeiras

create index "con-num_ser_eqp-sk"
        on contrato(num_ser_eqp);
create index "con-cod_pla-sk"
        on contrato(cod_pla);
create index "end-cod_usu-sk"
        on endereco(cod_usu);
create index "con-cod_usu-sk"
        on contrato(cod_usu);
create index "ate-cod_fun-sk"
        on atendimento(cod_fun);
create index "ate-cod_con-sk"
        on atendimento(cod_con);
create index "fat-cod_con-sk"
        on fatura(cod_con);
create index "usu-cod_fun-sk"
        on usuario(cod_fun);

-- Criação dos índices nas colunas mais utilizadas

create index "usu-cpf_cnpj_usu-sk"
        on usuario(cpf_cnpj_usu);
create index "usu-tel_usu-sk"
        on usuario(tel_usu);
create index "ate-obs_ate-sk"
        on atendimento(obs_ate);
create index "equ-nom_equ-sk"
        on equipamento(nom_equ);
create index "pla-vel_pla-sk"
        on plano(vel_pla);
```

APÊNDICE B - Script da criação dos privilégios de usuários

```
-- Criação das roles
create role comercial;
create role suporte;
create role financeiro;
create role administrador;

-- Atribuindo permissoes para Comercial
grant select,insert,update on endereco,usuario,contrato to comercial with grant option;
grant select on atendimento, fatura, plano to comercial with grant option;
-- Atribuindo permissoes para Suporte
grant select, update on endereco, usuario to suporte with grant option;
grant select on atendimento, contrato, fatura, plano to suporte with grant option;
-- Atribuindo permissoes para Financeiro
grant select, insert, update on fatura to financeiro with grant option;
grant select on endereco, usuario, funcionario, atendimento, contrato, plano to financeiro with grant option;
-- Atribuindo permissoes para Administrador
grant select, delete, insert, update on endereco, usuario, funcionario, atendimento, contrato, fatura, plano to administrador;

-- Criando acessos para os usuarios do sistema
create role joao password '111' in role comercial;
create role lucas password '222' in role suporte;
create role jonas password '333' in role financeiro;
create role wiliam password '444' in role administrador;

-- Atribuindo permissão para logar com o usuario
alter role joao LOGIN;
alter role lucas LOGIN;
alter role jonas LOGIN;
alter role wiliam LOGIN;
```

APÊNDICE C - *Script* de criação da *trigger* e *function* “verificaFatura”

```
CREATE FUNCTION verificaFatura() RETURNS trigger
AS $$
   BEGIN

      IF NEW.cod_fat EXISTS THEN
         RAISE EXCEPTION 'Codigo Da Fatura Ja Existe';
      END IF;
      IF NEW.qtd_tot_dds is NULL THEN
         RAISE EXCEPTION 'Quantidade nao pode ser nula';
      END IF;
      IF NEW.qtd_tot_dds < 1 THEN
         RAISE EXCEPTION 'Nao e possivel faturar clientes sem consumo de banda';
      END IF;

   END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER verificaFatura BEFORE INSERT OR UPDATE ON fatura
   FOR EACH ROW EXECUTE FUNCTION verificaFatura();
```

APÊNDICE D - *Scripts* de criação da auditoria sobre a tabela contrato

```
-- Criação da trigger para auditoria da tabela contrato

-- Criando a tabela auditoria de contrato
create table contrato_auditoria (
	cod_con numeric(5) not null,
	dat_ini_con date not null,
	dat_fin_con date not null,
	cod_usu numeric (5) not null,
	cod_pla numeric (2) not null,
	num_ser_eqp numeric (8) not null,
	dat_alt Date not null
);

-- Criação da trigger
create or replace trigger contrato_auditoria_trigger
	after insert or delete on contrato
	for each row
	execute procedure contrato_auditoria_function();

-- Criação da procedure
create or replace function contrato_auditoria_function ()
	returns trigger as
		$$
			begin
				insert into contrato_auditoria
				(cod_con, dat_ini_con, dat_fin_con, cod_usu, cod_pla,
num_ser_eqp, dat_alt)
					values (new.cod_con, new.dat_ini_con,
new.dat_fin_con, new.cod_usu, new.cod_pla, new.num_ser_eqp, current_date);
				return new;
			end
		$$
	language plpgsql;
```

APÊNDICE E - *Script* de criação da *function* para consultar usuário clientes

```
-- Função para filtrar apenas usuários que são clientes

CREATE OR REPLACE FUNCTION getClientes() RETURNS SETOF usuario AS
$body$
DECLARE
 x usuario%rowtype;
BEGIN
        FOR x IN SELECT * FROM usuario

 LOOP
        IF (x.cod_fun is null) THEN
                RETURN NEXT x;
        END IF;
 END LOOP;

 RETURN;
END
$body$
LANGUAGE plpgsql;
```

APÊNDICE F - *Script* de criação da *function* para consultar os atendimentos pelo CPF ou CNPJ do usuário

```
-- Função para procurar todos os atendimentos de um usuário a partir do CPF ou CNPJ deste

CREATE OR REPLACE FUNCTION getUsuarioAtendimento(cpf numeric(14))
RETURNS table(cod_ate numeric, dat_ate date, obs_ate varchar) AS
$body$
BEGIN
 RETURN query select a.cod_ate, a.dat_ate, a.obs_ate from usuario u
        inner join contrato c on u.cod_usu = c.cod_usu
        inner join atendimento a on c.cod_con = a.cod_con
        where u.cpf_cnpj_usu  = $1
        order by u.cod_usu;
END
$body$
LANGUAGE 'plpgsql';

-- Chamando a função. Alternar "X" pelo CPF ou CNPJ do usuário
select * from getusuarioatendimento(x);
```

APÊNDICE G - *Script* de criação das *views* com base nas consultas solicitadas

- Script para criação de view, a partir do primeiro relatório solicitado.

```
/* PRIMEIRO RELATÓRIO
Relacionar o código, nome e todos os clientes que são pessoa física.
Ordene o relatório de forma descendente pelo nome.
*/
create or replace view vw_relatorio1 as
select cod_usu "Código do usuário", nom_usu "Nome do usuário", cpf_cnpj_usu "CPF do usuário"
from usuario u
where length (cpf_cnpj_usu::varchar) < 13
order by nom_usu desc;
```

```
-- Script para criação de view, a partir do segundo relatório solicitado.
/* SEGUNDO RELATÓRIO
Relacionar o nome do cliente, nome do plano e a velocidade para todos os clientes.
Filtre somente clientes com planos com velocidades maiores que 100MB.
Ordene o relatório de forma descendente pelo nome do plano.
*/
create or replace view vw_relatorio2 as
select u.nom_usu "Nome do usuário", p.nom_pla "Nome do plano", p.vel_pla "Velocidade do plano"
        from usuario u
        inner join contrato c on u.cod_usu = c.cod_usu
        inner join plano p on c.cod_pla = p.cod_pla
        where p.vel_pla > 100
        order by p.nom_pla desc;
```

```
-- Script para criação de view, a partir do terceiro relatório solicitado.
/* TERCEIRO RELATÓRIO
Relacionar o código do cliente, nome do cliente, quantidade total de
atendimentos nos meses pares de 2022. Ordene o relatório do cliente com mais
atendimentos(em termos de quantidade) para o cliente com menos atendimentos.
*/
```

```
create or replace view vw_relatorio3 as
select u.cod_usu "Código do usuário", u.nom_usu "Nome do usuário", count(a.cod_ate) "Quantidade de atendimentos"
        from usuario u
        inner join contrato c on u.cod_usu = c.cod_usu
        inner join atendimento a on c.cod_con = a.cod_con
        where mod(extract(month from a.dat_ate), 2) = 0
        group by u.cod_usu
        order by 3 desc;

-- Script para criação de view, a partir do quarto relatório solicitado.
/* QUARTO RELATÓRIO
Relacionar o cpf do cliente, nome do cliente e o valor total de uso da internet.
Filtrar somente clientes do sexo masculino, com idades pares e que realizaram compras em
meses entre 01 e 08 de 2022. Ordene o relatório do cliente com maior uso até o cliente com menor uso.
*/
create or replace view vw_relatorio4 as
select u.cpf_cnpj_usu "CPF do usuário", u.nom_usu "Nome do usuário", f.qtd_tot_dds "Quantidade de dados"
        from usuario u
        inner join contrato c on u.cod_usu = c.cod_usu
        inner join fatura f on c.cod_con = f.cod_con
        where u.sex_usu = 'M' and mod(extract(year from age(u.dat_nas_usu)), 2) = 0
        and extract(month from c.dat_ini_con) > 0 and extract(month from c.dat_ini_con) < 9
        and extract(year from c.dat_ini_con) = 2022
        order by f.qtd_tot_dds desc;
```

APÊNDICE H - *Script* para criação do backup do banco de dados

```
@ECHO off
TITLE InsideProvider
cls
color a


:MenuInicial
cls
ECHO.
ECHO    ----------------------------------------
ECHO    -- * MENU BACKUP INSIDEPROVIDER  * --
ECHO    -- *   ESCOLHA UMA OPCAO ABAIXO    * --
ECHO    ----------------------------------------
ECHO.
ECHO    [1] BACKUP
ECHO    [X] Sair
ECHO.
set /p MenuInicial=
if '%MenuInicial%'=='1' goto MenuInicial_backup
if '%MenuInicial%'=='x' goto sair_MenuInicial
if not %MenuInicial%=="1,X" (
ECHO    Opcao [%MenuInicial%] invalida.
ECHO    Verifique o menu acima.

goto MenuInicial
)

:MenuInicial_backup
MKDIR C:\Backup_InsideProvider
cd C:\Backup_InsideProvider
ECHO    %date:~6,4%.%date:~3,2%.%date:~0,2%    -    %time:~0,2%:%time:~-8,2%:
BACKUP iniciado >> sgbd.log
ECHO Em execucao...
```

```
"c:\Program Files\PostgreSQL\14\bin\pg_dump.exe" -Fc -U postgres -d inside_provider_db -C -Fp -f  backup.bak >> sgbd.log
ren "C:\Backup_InsideProvider\backup.bak" "Backup_insideprovider_%date:~6,4%.%date:~3,2%.%date:~0,2%_%time:~0,2%%time:~-8,2%.bak" >> sgbd.log

ECHO %date:~6,4%.%date:~3,2%.%date:~0,2% - %time:~0,2%:%time:~-8,2%: BACKUP concluido >> sgbd.log
goto MenuInicial
```

APÊNDICE I - *Script* para criação do *restore* do banco de dados

```
@ECHO off
TITLE InsideProvider
cls
color a


:MenuInicial
cls
ECHO.
ECHO    ----------------------------------------
ECHO    -- * MENU RESTORE INSIDEPROVIDER  * --
ECHO    -- *   ESCOLHA UMA OPCAO ABAIXO    * --
ECHO    ----------------------------------------
ECHO.
ECHO    [1] RESTORE
ECHO    [X] Sair
ECHO.
set /p MenuInicial=
if '%MenuInicial%'=='1' goto MenuInicial_restore
if '%MenuInicial%'=='x' goto sair_MenuInicial
if '%MenuInicial%'=='X' goto sair_MenuInicial
if not %MenuInicial%=="1,x,X" (
ECHO    Opcao [%MenuInicial%] invalida.
ECHO    Verifique o menu acima.
goto MenuInicial
)
:MenuInicial_restore
cd C:\Backup_InsideProvider
cls
ECHO O nome da base restaurada deve ser "Backup.bak". >> sgbd.log
:Menu_restore
ECHO.
color 4
```

```
ECHO    ------------------------------------------------------
ECHO    -- *              LEIA COM ATENCAO!                * --
timeout 4 > NUL
color 0B
ECHO    -- *      O NOME DO ARQUIVO DEVE SER backup.bak     * --
ECHO    -- *   O ARQUIVO DEVE ESTAR EM C:\Backup_InsideProvider   * --
ECHO    ------------------------------------------------------
ECHO.
ECHO    [1] PROSSEGUIR
ECHO    [X] SAIR
ECHO.
set /p Menu_restore=
if '%Menu_restore%'=='1' goto Restore_prosseguir
if '%Menu_restore%'=='x' goto MenuInicial
if '%Menu_restore%'=='X' goto MenuInicial
:Restore_prosseguir
cd C:\Backup_InsideProvider
ECHO.
ECHO Em execucao o processo de restauracao...
ECHO.
ECHO    %date:~6,4%.%date:~3,2%.%date:~0,2%    -    %time:~0,2%:%time:~-8,2%:
RESTORE iniciado >> sgbd.log
ECHO Informe a senha para apagar a base de dados existente:
"c:\Program Files\PostgreSQL\14\bin\dropdb.exe" -U postgres inside_provider_db
ECHO.
ECHO Informe a senha para restaurar o backup...
"c:\Program    Files\PostgreSQL\14\bin\psql.exe"    -U    postgres    -d    postgres    <
"backup.bak" >> sgbd.log
ECHO    %date:~6,4%.%date:~3,2%.%date:~0,2%    -    %time:~0,2%:%time:~-8,2%:
RESTORE concluido >> sgbd.log
ECHO %date:~6,4%.%date:~3,2%.%date:~0,2% - %time:~0,2%:%time:~-8,2%: Base
restaurada >> sgbd.log

goto MenuInicial
```

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS**. NBR 14724: Informação e documentação – Trabalhos acadêmicos – Apresentação**. Rio de Janeiro, 2011.

MOREIRA, Flávio Ferry de Oliveira. **Fundamentos de banco de dados**. 2013. 102p. Universidade Federal do Piauí – UFPI, 2013.

7GRAUS. **Ferramentas Online Grátis**. 2022. Disponível em:

https://www.4devs.com.br/. Acesso em: 25 jun. 2022.