UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

INSTITUTO METRÓPOLE DIGITAL

BANCO DE DADOS

CRIAÇÃO DE BANCO DE DADOS  
CLÍNICA DENTÁRIA

DÉBORA EMILI COSTA OLIVEIRA  
JOSÉ GAMELEIRA DO RÊGO NETO

Sumário

[**1.** OBJETIVO 3](#_Toc436042276)

[**2.** INTRODUÇÃO 4](#_Toc436042277)

[**3.** DIAGRAMA (DER) 5](#_Toc436042279)

[**4.** SCRIPT DE CRIAÇÃO](#_Toc436042287) 6

[**5.** CONLUSÃO 17](#_Toc436042288)

1. OBJETIVO

O trabalho final da disciplina de Banco de Dados consiste em aplicar e fixar os conceitos aprendido em sala de aula sobre a definição, a manipulação e a consulta no banco de dados. Ter contato com a plataforma SQL PostgreeSQL, a qual é uma plataforma grátis, de código aberto, e é considerada a mais robusta nessa modalidade. E além do mais obter a nota referente à terceira unidade da disciplina de Banco de Dados.

1. INTRODUÇÃO

O projeto final da disciplina de Banco de Dados (BD) consiste em desenvolver um banco de dados, como também, um conjunto instruções que façam com que este seja mais robusto e apresente ferramentas que evitem inconsistências das informações armazenadas. Servindo para comprovação da obtenção do conhecimento determinado no plano da disciplina.

A aplicação prática do trabalho é focado na construção de uma base de dados que possam ser utilizadas por uma aplicação para clínicas odontológicas. Buscou-se construir um conjunto de informações que possam englobar de maneira satisfatória, as operações básicas e serviços prestados. Dessa maneira, o cliente tem uma ferramenta satisfatória para se construir uma aplicação que use essa base de dados.

Fisicamente, a base de dados apresenta tabelas, índices, funções e gatilhos que facilitem a manipulação correta das informações. Obteve-se o cuidado de fazer com que operações e modificações que causariam inconsistência, não fossem permitidas.

1. METODOLOGIA

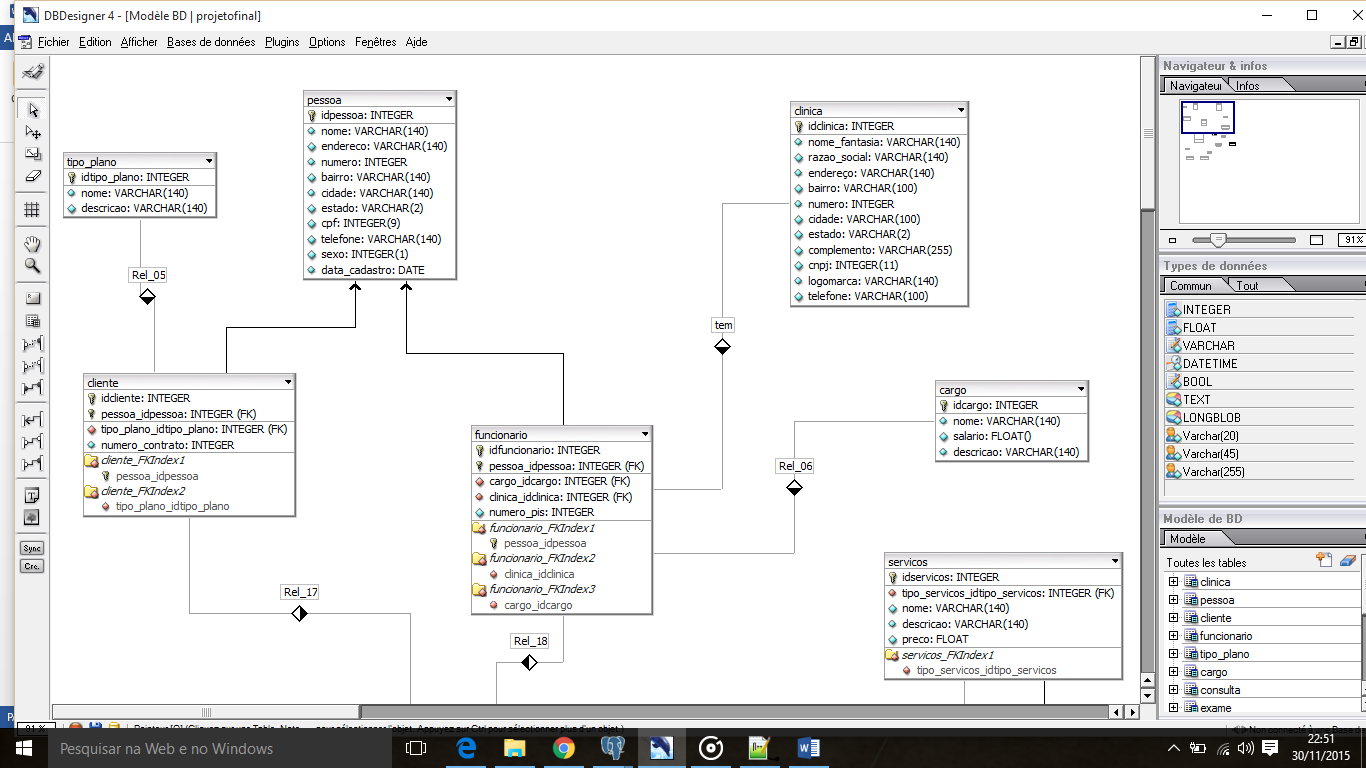
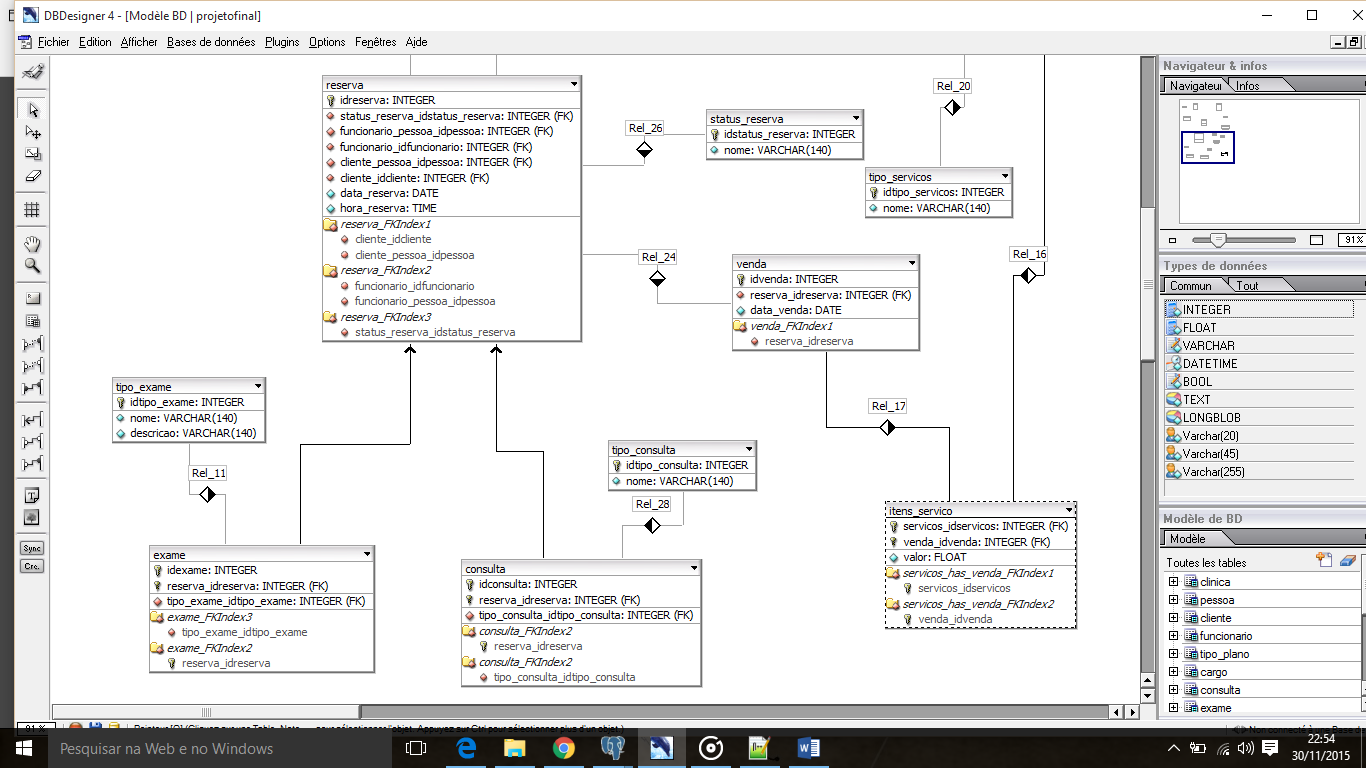
Para se desenvolver este projeto foi utilizado ferramentas de caráter profissional para se manter um bom nível. Para construção do Diagrama Entidade-Relacional (DER), foi utilizado o DBDesiner.

O diagrama DER é uma opção bastante utilizada para se ter uma ideia mais concreta de como se relacionaram as entidades do projeto. Todas as tabelas e seus relacionamentos são representados por símbolos que passam ao usuário a ideia principal de como elas serão e como cada um se comunica e coexiste com outra tabela.

Para criação do código do banco foi utilizada a linguagem SQL para o PostgreSQL. A linguagem SQL é uma linguagem bastante utilizada em banco de dados relacional. Consiste em uma linguagem baseada da álgebra relacional, o que facilita o entendimento para quem está tentando aprender a como se definir a forma com que o banco de dados se organiza.

O PostgresSQL apresenta uma ferramenta que facilita a visualização e a inserção de novos comandos, esta é o pgAdmin III. O pgAdmin III foi usado para se inserir as instruções SQL, servindo como uma ferramenta gráfica de administração no banco. Nele é possível, visualizar a maneira como os dados estão sendo guardados logicamente nas tabelas.

Para gerar o DER, foi pensado em todas as necessidades básicas de uma clínica odontológicas, abrangendo todas as funcionalidades mais triviais.

1. DIAGRAMA (DER)
2. SCRIPT DE CRIAÇÃO

-- Criando o banco

CREATE DATABASE clinicadentaria

WITH OWNER = postgres

ENCODING = 'UTF8'

TABLESPACE = pg\_default

LC\_COLLATE = 'Portuguese\_Brazil.1252'

LC\_CTYPE = 'Portuguese\_Brazil.1252'

CONNECTION LIMIT = -1;

----------------------------------------------------------------------------------------------

--TABELAS

----------------------------------------------------------------------------------------------

CREATE TABLE cargo (

idcargo SERIAL NOT NULL ,

nome VARCHAR(140) NOT NULL,

salario numeric(10,2),

descricao VARCHAR(140) NULL,

PRIMARY KEY(idcargo)

);

CREATE TABLE clinica (

idclinica SERIAL NOT NULL,

nome\_fantasia VARCHAR(140) NOT NULL,

razao\_social VARCHAR(140) NOT NULL,

endereço VARCHAR(140) NOT NULL,

bairro VARCHAR(100) NOT NULL,

numero INTEGER NOT NULL,

cidade VARCHAR(100) NOT NULL,

estado VARCHAR(2) NOT NULL,

complemento VARCHAR(255) NULL,

cnpj INTEGER NOT NULL,

logomarca VARCHAR(140) NOT NULL,

telefone VARCHAR(100) NOT NULL,

PRIMARY KEY(idclinica)

);

CREATE TABLE pessoa (

idpessoa SERIAL NOT NULL,

nome VARCHAR(140) NOT NULL,

endereco VARCHAR(140) NOT NULL,

numero INTEGER NOT NULL,

bairro VARCHAR(140) NOT NULL,

cidade VARCHAR(140) NOT NULL,

estado VARCHAR(2) NOT NULL,

cpf INTEGER NOT NULL,

telefone VARCHAR(140) NOT NULL,

sexo INTEGER NOT NULL,

data\_cadastro DATE NOT NULL,

PRIMARY KEY(idpessoa)

);

CREATE TABLE status\_reserva (

idstatus\_reserva SERIAL NOT NULL,

nome VARCHAR(140) NULL,

PRIMARY KEY(idstatus\_reserva)

);

CREATE TABLE tipo\_consulta (

idtipo\_consulta SERIAL NOT NULL,

nome VARCHAR(140) NULL,

PRIMARY KEY(idtipo\_consulta)

);

CREATE TABLE tipo\_exame (

idtipo\_exame SERIAL NOT NULL,

nome VARCHAR(140) NULL,

descricao VARCHAR(140) NULL,

PRIMARY KEY(idtipo\_exame)

);

CREATE TABLE tipo\_plano (

idtipo\_plano SERIAL NOT NULL,

nome VARCHAR(140) NOT NULL,

descricao VARCHAR(140) NULL,

PRIMARY KEY(idtipo\_plano)

);

CREATE TABLE tipo\_servicos

(

idtipo\_servicos serial NOT NULL,

nome character varying(140) NOT NULL,

CONSTRAINT tipo\_servicos\_pkey PRIMARY KEY (idtipo\_servicos)

);

CREATE TABLE servicos

(

idservicos serial NOT NULL,

nome character varying(140),

descricao character varying(140),

preco numeric(10,2),

tipo\_servicos\_idtipo\_servicos integer,

CONSTRAINT servicos\_pkey PRIMARY KEY (idservicos),

CONSTRAINT "servicos\_FKIndextipo\_servicos" FOREIGN KEY (tipo\_servicos\_idtipo\_servicos)

REFERENCES tipo\_servicos (idtipo\_servicos) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE cliente

(

idcliente serial NOT NULL,

pessoa\_idpessoa integer NOT NULL,

tipo\_plano\_idtipo\_plano integer NOT NULL,

numero\_contrato integer NOT NULL,

CONSTRAINT cliente\_pkey PRIMARY KEY (idcliente, pessoa\_idpessoa),

CONSTRAINT "cliente\_FKIndexpessoa" FOREIGN KEY (pessoa\_idpessoa)

REFERENCES pessoa (idpessoa) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "cliente\_FKIndextiop\_plano" FOREIGN KEY (tipo\_plano\_idtipo\_plano)

REFERENCES tipo\_plano (idtipo\_plano) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE funcionario

(

idfuncionario serial NOT NULL,

pessoa\_idpessoa integer NOT NULL,

cargo\_idcargo integer NOT NULL,

clinica\_idclinica integer NOT NULL,

numero\_pis integer NOT NULL,

CONSTRAINT funcionario\_pkey PRIMARY KEY (idfuncionario, pessoa\_idpessoa),

CONSTRAINT "funcionario\_FKIndexcargo" FOREIGN KEY (cargo\_idcargo)

REFERENCES cargo (idcargo) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "funcionario\_FKIndexclinica" FOREIGN KEY (clinica\_idclinica)

REFERENCES clinica (idclinica) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "funcionario\_FKIndexpessoa" FOREIGN KEY (pessoa\_idpessoa)

REFERENCES pessoa (idpessoa) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE reserva

(

idreserva serial NOT NULL,

status\_reserva\_idstatus\_reserva integer NOT NULL,

funcionario\_pessoa\_idpessoa integer NOT NULL,

funcionario\_idfuncionario integer NOT NULL,

cliente\_pessoa\_idpessoa integer NOT NULL,

cliente\_idcliente integer NOT NULL,

data\_reserva date,

hora\_reserva time without time zone,

CONSTRAINT reserva\_pkey PRIMARY KEY (idreserva),

CONSTRAINT "reserva\_FKIndexcliente" FOREIGN KEY (cliente\_idcliente, cliente\_pessoa\_idpessoa)

REFERENCES cliente (idcliente, pessoa\_idpessoa) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "reserva\_FKIndexfuncionario" FOREIGN KEY (funcionario\_idfuncionario, funcionario\_pessoa\_idpessoa)

REFERENCES funcionario (idfuncionario, pessoa\_idpessoa) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "reserva\_FKIndextipo\_reserva" FOREIGN KEY (status\_reserva\_idstatus\_reserva)

REFERENCES status\_reserva (idstatus\_reserva) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE consulta

(

idconsulta serial NOT NULL,

reserva\_idreserva integer NOT NULL,

tipo\_consulta\_idtipo\_consulta integer NOT NULL,

CONSTRAINT consulta\_pkey PRIMARY KEY (idconsulta, reserva\_idreserva),

CONSTRAINT "consulta\_FKIndexreserva" FOREIGN KEY (reserva\_idreserva)

REFERENCES reserva (idreserva) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "consulta\_FKIndextipo\_consulta" FOREIGN KEY (tipo\_consulta\_idtipo\_consulta)

REFERENCES tipo\_consulta (idtipo\_consulta) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE exame

(

idexame serial NOT NULL,

reserva\_idreserva integer NOT NULL,

tipo\_exame\_idtipo\_exame integer NOT NULL,

CONSTRAINT exame\_pkey PRIMARY KEY (idexame, reserva\_idreserva),

CONSTRAINT "exame\_FKIndexreserva" FOREIGN KEY (reserva\_idreserva)

REFERENCES reserva (idreserva) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "exame\_FKIndextipo\_exame" FOREIGN KEY (tipo\_exame\_idtipo\_exame)

REFERENCES tipo\_exame (idtipo\_exame) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE venda

(

idvenda serial NOT NULL,

reserva\_idreserva integer NOT NULL,

data\_venda date,

CONSTRAINT venda\_pkey PRIMARY KEY (idvenda),

CONSTRAINT "venda\_FKIndexreserva" FOREIGN KEY (reserva\_idreserva)

REFERENCES reserva (idreserva) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

CREATE TABLE itens\_servico

(

servicos\_idservicos integer NOT NULL,

venda\_idvenda integer NOT NULL,

valor numeric(10,2),

CONSTRAINT itens\_servico\_pkey PRIMARY KEY (servicos\_idservicos, venda\_idvenda),

CONSTRAINT "itens\_servicoFKIndexvenda" FOREIGN KEY (venda\_idvenda)

REFERENCES venda (idvenda) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT "itens\_servicosFKIndexservico" FOREIGN KEY (servicos\_idservicos)

REFERENCES servicos (idservicos) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

);

----------------------------------------------------------------------------------------------

--INDICES

----------------------------------------------------------------------------------------------

CREATE INDEX idcargo\_idx ON cargo (idcargo);

CREATE INDEX idcliente\_idx ON cliente (idcliente, pessoa\_idpessoa);

CREATE INDEX idclinica\_idx ON clinica (idclinica);

CREATE INDEX idconsulta\_idx ON consulta (idconsulta, reserva\_idreserva);

CREATE INDEX idexame\_idx ON exame (idexame, reserva\_idreserva);

CREATE INDEX idfuncionario\_idx ON funcionario (idfuncionario, pessoa\_idpessoa);

CREATE INDEX iditens\_servico\_idx ON itens\_servico (servicos\_idservicos, venda\_idvenda);

CREATE INDEX idpessoa\_idx ON pessoa (idpessoa);

CREATE INDEX idreserva\_idx ON reserva (idreserva);

CREATE INDEX idservicos\_idx ON servicos (idservicos);

CREATE INDEX idstatus\_reserva\_idx ON status\_reserva (idstatus\_reserva);

CREATE INDEX idtipo\_consulta\_idx ON tipo\_consulta (idtipo\_consulta);

CREATE INDEX idtipo\_exame\_idx ON tipo\_exame (idtipo\_exame);

CREATE INDEX idtipo\_plano\_idx ON tipo\_plano (idtipo\_plano);

CREATE INDEX idtipo\_servicos\_idx ON tipo\_servicos (idtipo\_servicos);

CREATE INDEX idvenda\_idx ON venda (idvenda);

----------------------------------------------------------------------------------------------

--FUNCOES

----------------------------------------------------------------------------------------------

--Verifica se o usuário tenta inserir/atualizar o salário com um valor negativo

CREATE FUNCTION cargo\_salario() RETURNS trigger AS $cargo\_salario$

BEGIN

IF NEW.salario < 0 THEN

RAISE EXCEPTION 'O salário não pode ser negativo!';

END IF;

RETURN NEW;

END;

$cargo\_salario$ LANGUAGE plpgsql;

CREATE TRIGGER cargo\_salario BEFORE INSERT OR UPDATE

ON cargo

FOR EACH ROW EXECUTE

PROCEDURE cargo\_salario();

--Verifica se o usuário tenta inserir/atualizar o contrato com um valor negativo ou zerado

CREATE FUNCTION cliente\_numerocontrato() RETURNS trigger AS $cliente\_numerocontrato$

BEGIN

IF NEW.numero\_contrato <= 0 THEN

RAISE EXCEPTION 'O número do contrato não pode ser negativo!';

END IF;

RETURN NEW;

END;

$cliente\_numerocontrato$ LANGUAGE plpgsql;

CREATE TRIGGER cliente\_numerocontrato BEFORE INSERT OR UPDATE

ON cliente

FOR EACH ROW EXECUTE

PROCEDURE cliente\_numerocontrato();

--Verifica se o usuário tenta inserir/atualizar o número do endereco com um valor negativo

CREATE FUNCTION clinica\_numero() RETURNS trigger AS $clinica\_numero$

BEGIN

IF NEW.numero < 0 THEN

RAISE EXCEPTION 'O número não pode ser negativo!';

END IF;

RETURN NEW;

END;

$clinica\_numero$ LANGUAGE plpgsql;

CREATE TRIGGER clinica\_numero BEFORE INSERT OR UPDATE

ON clinica

FOR EACH ROW EXECUTE

PROCEDURE clinica\_numero();

--Verifica se o usuário tenta inserir/atualizar o número do PIS com um valor negativo ou zerado

CREATE FUNCTION funcionario\_numeropis() RETURNS trigger AS $funcionario\_numeropis$

BEGIN

IF NEW.numero\_pis <= 0 THEN

RAISE EXCEPTION 'O numero de pis não pode ser negativo!';

END IF;

RETURN NEW;

END;

$funcionario\_numeropis$ LANGUAGE plpgsql;

CREATE TRIGGER funcionario\_numeropis BEFORE INSERT OR UPDATE

ON funcionario

FOR EACH ROW EXECUTE

PROCEDURE funcionario\_numeropis();

--Verifica se o usuário tenta inserir/atualizar o valor do item de serviço com um valor negativo

CREATE FUNCTION itensservico\_valor() RETURNS trigger AS $itensservico\_valor$

BEGIN

IF NEW.valor < 0 THEN

RAISE EXCEPTION 'O valor do item não pode ser negativo!';

END IF;

RETURN NEW;

END;

$itensservico\_valor$ LANGUAGE plpgsql;

CREATE TRIGGER itensservico\_valor BEFORE INSERT OR UPDATE

ON itens\_servico

FOR EACH ROW EXECUTE

PROCEDURE itensservico\_valor();

--Verifica se o usuário tenta inserir/atualizar o número do endereco da pessoa com um valor negativo

CREATE FUNCTION pessoa\_numero() RETURNS trigger AS $pessoa\_numero$

BEGIN

IF NEW.numero < 0 THEN

RAISE EXCEPTION 'O numero não pode ser negativo!';

END IF;

RETURN NEW;

END;

$pessoa\_numero$ LANGUAGE plpgsql;

CREATE TRIGGER pessoa\_numero BEFORE INSERT OR UPDATE

ON pessoa

FOR EACH ROW EXECUTE

PROCEDURE pessoa\_numero();

--Verifica se o usuário tenta inserir/atualizar o preço do serviço com um valor negativo

CREATE FUNCTION servicos\_preco() RETURNS trigger AS $servicos\_preco$

BEGIN

IF NEW.preco < 0 THEN

RAISE EXCEPTION 'O preço não pode ser negativo!';

END IF;

RETURN NEW;

END;

$servicos\_preco$ LANGUAGE plpgsql;

CREATE TRIGGER servicos\_preco BEFORE INSERT OR UPDATE

ON servicos

FOR EACH ROW EXECUTE

PROCEDURE servicos\_preco();

--Verifica se o usuário tenta inserir/atualizar a data de venda para uma data maior do que a data atual

CREATE FUNCTION venda\_data() RETURNS trigger AS $venda\_data$

BEGIN

IF NEW.data\_venda > current\_timestamp::date THEN

RAISE EXCEPTION 'A data da venda não pode ser maior que a data de hoje';

END IF;

RETURN NEW;

END;

$venda\_data$ LANGUAGE plpgsql;

CREATE TRIGGER venda\_data BEFORE INSERT OR UPDATE

ON venda

FOR EACH ROW EXECUTE

PROCEDURE venda\_data();

--Verifica se o usuário tenta inserir/atualizar a data de venda para uma data menor do que a data atual

CREATE FUNCTION reserva\_data() RETURNS trigger AS $reserva\_data$

BEGIN

IF NEW.data\_reserva <= current\_timestamp::date THEN

RAISE EXCEPTION 'A data da reserva não pode ser menor ou igual a data de hoje';

END IF;

RETURN NEW;

END;

$reserva\_data$ LANGUAGE plpgsql;

CREATE TRIGGER reserva\_data BEFORE INSERT OR UPDATE

ON reserva

FOR EACH ROW EXECUTE

PROCEDURE reserva\_data();

--Verifica se o usuário tenta inserir/atualizar um número de cpf que tenha número menor ou maior que 11 digitos

CREATE FUNCTION pessoa\_cpf() RETURNS trigger AS $pessoa\_cpf$

BEGIN

IF length(NEW.cpf::character varying) <> 11 THEN

RAISE EXCEPTION 'O cpf tem que ter 11 numeros';

END IF;

RETURN NEW;

END;

$pessoa\_cpf$ LANGUAGE plpgsql;

CREATE TRIGGER pessoa\_cpf BEFORE INSERT OR UPDATE

ON pessoa

FOR EACH ROW EXECUTE

PROCEDURE pessoa\_cpf();

--Verifica se o usuário tenta inserir/atualizar um número de cnpj que tenha número menor ou maior que 14 digitos

CREATE FUNCTION clinica\_cnpj() RETURNS trigger AS $clinica\_cnpj$

BEGIN

IF length(NEW.cnpj::character varying) <> 14 THEN

RAISE EXCEPTION 'O cnpj tem que ter 14 numeros';

END IF;

RETURN NEW;

END;

$clinica\_cnpj$ LANGUAGE plpgsql;

CREATE TRIGGER clinica\_cnpj BEFORE INSERT OR UPDATE

ON clinica

FOR EACH ROW EXECUTE

PROCEDURE clinica\_cnpj();

1. CONCLUSÃO

O projeto final da disciplina de Banco de Dados serviu de maneira satisfatória fazer com que os alunos consolidassem os conhecimentos adquiridos no decorrer da disciplina, construindo um saber científico que será necessário no decorrer da vida acadêmica. Ao termino desse projeto, foi possível conseguir sentir as dificuldades da construção de um banco de dados, desde a projeção dele a partir de um DER até a construção de funções e gatilhos que impedem modificações dos dados de maneira equivocadas.

Como foi possível perceber, o trabalho possibilitou uma excelente experiência acadêmica e percepção de como resolver um problema que pode acontecer no mercado de trabalho. Isso é fundamental para criação de um profissional capacitado, que esteja pronto para o mercado de trabalho e que tenha condições de crescimento acadêmico