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

NATAL, 2015

Sumário

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.

2. 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.

3. 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.

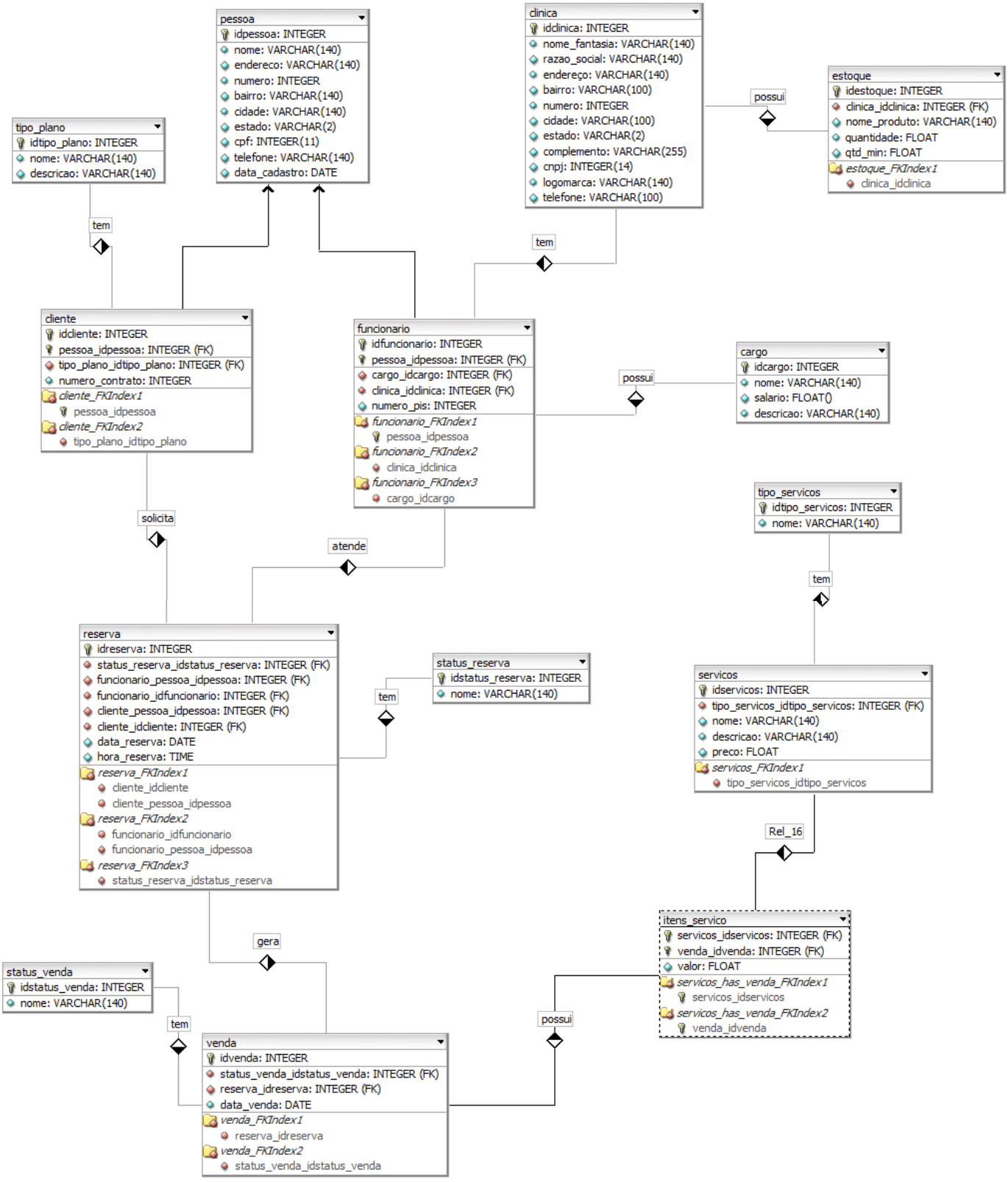
O diagrama DER é uma opção bastante utilizada para se ter uma ideia mais concreta de como se relacionaram as entidades do projeto. Todas as tabelas e seus relacionamentos são representados por símbolos que passam ao usuário a ideia principal de como elas serão e como cada um se comunica e coexiste com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

4. DIAGRAMA (DER)



5. 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,
  endereco 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 bigint 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 bigint  NOT NULL,
  telefone VARCHAR(140) NOT NULL,
  data_cadastro DATE NOT NULL,
  PRIMARY KEY(idpessoa),
  CONSTRAINT pessoa_cpf_key UNIQUE (cpf)
);

CREATE TABLE status_reserva (
```

```
  idstatus_reserva SERIAL NOT NULL,
  nome VARCHAR(140) NULL,
  PRIMARY KEY(idstatus_reserva)
);

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 status_venda
(
  idstatus_venda serial NOT NULL,
  nome character varying(140) NOT NULL,
  CONSTRAINT pk_status_venda PRIMARY KEY (idstatus_venda)
);

CREATE TABLE venda
(
  idvenda serial NOT NULL,
  reserva_idreserva integer NOT NULL,
  data_venda date,
  status_venda_idstatus_venda integer NOT NULL,
  CONSTRAINT venda_pkey PRIMARY KEY (idvenda),
  CONSTRAINT "venda_FKIndexreserva" FOREIGN KEY (reserva_idreserva)
    REFERENCES reserva (idreserva) MATCH SIMPLE
    ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT "venda_FKIndexstatus" FOREIGN KEY (status_venda_idstatus_venda)
    REFERENCES status_venda (idstatus_venda) 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
);

CREATE TABLE estoque
(
  idestoque serial NOT NULL,
  nome_produto character varying(140) NOT NULL,
  quantidade numeric(10,2) NOT NULL,
  qtd_minimo numeric(10,2) NOT NULL,
  clinica_idclinica integer NOT NULL,
  CONSTRAINT estoque_pkey PRIMARY KEY (idestoque),
  CONSTRAINT "estoque_FKIndexclinica" FOREIGN KEY (clinica_idclinica)
      REFERENCES clinica (idclinica) 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 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 idstatus_venda_idx ON status_venda (idstatus_venda);

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);

CREATE INDEX idestoque_idx ON estoque (idestoque);


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

--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();


CREATE FUNCTION reserva_venda() RETURNS trigger AS $reserva_venda$
BEGIN
IF (TG_OP = 'INSERT') THEN
        INSERT INTO venda (reserva_idreserva, status_venda_idstatus_venda) VALUES
(NEW.idreserva, 1);
        RETURN NEW;

    END IF;
    RETURN NULL; -- result is ignored since this is an AFTER trigger
  END;
$reserva_venda$ LANGUAGE plpgsql;

CREATE TRIGGER reserva_venda AFTER INSERT OR UPDATE OR DELETE
ON reserva
FOR EACH ROW EXECUTE
PROCEDURE reserva_venda();

--Verifica se o usuário tenta inserir/atualizar a quantidade com um valor negativo
CREATE FUNCTION estoque_quantidade() RETURNS trigger AS $estoque_quantidade$
BEGIN
IF NEW.quantidade < 0 THEN
RAISE EXCEPTION 'A quantidade não pode ser negativa!';
END IF;
RETURN NEW;
END;
$estoque_quantidade$ LANGUAGE plpgsql;

CREATE TRIGGER estoque_quantidade  BEFORE INSERT OR UPDATE
ON estoque
FOR EACH ROW EXECUTE
PROCEDURE estoque_quantidade();
```

```
--Verifica se o usuário tenta inserir/atualizar a qtd_minima com um valor negativo
CREATE FUNCTION estoque_qtdminima() RETURNS trigger AS $estoque_qtdminima$
BEGIN
IF NEW.quantidade < 0 THEN
RAISE EXCEPTION 'A qtd_minima não pode ser negativa!';
END IF;
RETURN NEW;
END;
$estoque_qtdminima$ LANGUAGE plpgsql;

CREATE TRIGGER estoque_qtdminima BEFORE INSERT OR UPDATE
ON estoque
FOR EACH ROW EXECUTE
PROCEDURE estoque_qtdminima();
```

```
--POPULANDO O BANCO

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

INSERT INTO clinica VALUES (1,'Clinica Dentaria 1','Clinica Dentária LTDA','Rua
Teste','Teste', 123, 'Cidade Teste', 'RN', '', 12345678901234, 'LOGO', '(84)99999-
9999/(84)9999-9999');
INSERT INTO clinica VALUES (2,'Clinica Dentaria 2','Clinica Dentária LTDA','Rua Teste
2','Teste 2', 123, 'Cidade Teste 2', 'RN', '', 12345678901234, 'LOGO', '(84)99999-
9999/(84)9999-9999');

INSERT INTO cargo VALUES (1, 'Dentista', 3000.00, 'o médico responsável');
INSERT INTO cargo VALUES (2, 'Auxiliar', 2000.00, 'Auxiliar do dentista');
INSERT INTO cargo VALUES (3, 'Recepcionista', 1500.00, 'recepciona as pessoas');


INSERT INTO pessoa VALUES (1, 'Maria josé', 'Rua Teste', 123, 'Teste', 'TESTE', 'RN',
12345678901, '9999-9999', '2015-12-01');
INSERT INTO pessoa VALUES (2, 'Dr João Marcos', 'Rua Teste', 123, 'Teste', 'TESTE', 'RN',
12345678902, '9999-9999', '2015-12-01');
INSERT INTO pessoa VALUES (3, 'Dr Mariana Felicia', 'Rua Teste', 123, 'Teste', 'TESTE', 'RN',
12345678903, '9999-9999', '2015-12-01');
INSERT INTO pessoa VALUES (4, 'Carla Peres', 'Rua Teste', 123, 'Teste', 'TESTE', 'RN',
12345678904, '9999-9999', '2015-12-01');
INSERT INTO pessoa VALUES (5, 'Roberto Carlos', 'Rua Teste', 123, 'Teste', 'TESTE', 'RN',
12345678905, '9999-9999', '2015-12-01');

INSERT INTO tipo_plano VALUES (1, 'FLEX', 'Plano para pobre');
INSERT INTO tipo_plano VALUES (2, 'MASTER', 'Plano para rico');

INSERT INTO cliente VALUES (1, 4, 1, 123);
INSERT INTO cliente VALUES (2,5,2,321);

INSERT INTO status_reserva VALUES (1, 'CADASTRADA');
INSERT INTO status_reserva VALUES (2, 'CANCELADA');
INSERT INTO status_reserva VALUES (3, 'REMARCADA');
INSERT INTO status_reserva VALUES (4, 'FINALIZADA');
```

```
INSERT INTO tipo_servicos VALUES (1, 'EXAME');
INSERT INTO tipo_servicos VALUES (2, 'CONSULTA');

INSERT INTO servicos VALUES (1, 'EXTRAÇÃO DENTARIA', 'Tira o dente fora', 100.00, 2);
INSERT INTO servicos VALUES (2, 'LIMPEZA DENTARIA', 'Limpa o dente ', 75.00, 2);
INSERT INTO servicos VALUES (3, 'RAIO-X', 'Tira uma chapa do dente', 30.00, 1);
INSERT INTO servicos VALUES (4, 'HEMOGRAMA', 'Exame de sangue', 20.00, 1);

INSERT INTO funcionario VALUES (1, 1, 3,1, 123);
INSERT INTO funcionario VALUES (2, 2, 1,2, 1234);
INSERT INTO funcionario VALUES (3, 3, 1,1, 1235);

INSERT INTO status_venda VALUES (1, 'CADASTRADA');
INSERT INTO status_venda VALUES (2, 'FINALIZADA');
INSERT INTO status_venda VALUES (3, 'ESTORNADA');

INSERT INTO reserva VALUES (1, 1, 3,3, 5, 2, '2015-12-06', '10:00:00');
SELECT * FROM venda

INSERT INTO itens_servico VALUES (2,1, 75.00);
INSERT INTO itens_servico VALUES (3,1, 30.00);

INSERT INTO estoque VALUES (1,'Bisturi', 3.00, 1.00, 1);
INSERT INTO estoque VALUES (2,'Bisturi', 5.00, 1.00, 2);
INSERT INTO estoque VALUES (3,'Gas do riso', 10.50, 10.00, 1);
INSERT INTO estoque VALUES (4,'Gas do riso', 9.50, 10.00, 2);
```

6. 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