INSTITUTO FEDERAL MINAS GERAIS (IFMG) - CAMPUS BAMBUÍ
Banco de Dados I
Prof. Marcos Roberto Ribeiro

Lista de Exercícios 11

Observações:

- As instruções SQL para criação dos bancos de dados estão disponíveis no *Arquivos Compartilhados* da página acadêmica;

Exercício 1:

Quais as vantagens dos procedimentos armazenados?

Exercício 2:

Crie uma função que receba um número entre 1 e 12 e retorne o respectivo mês por extenso. Crie outra função similar, mas que receba um argumento do tipo **DATE**. Dica: use as funções **EXTRACT()** ou **DATE_PART()**

Solução:

```
CREATE FUNCTION mes_extenso(mes INT) RETURNS TEXT AS $$
DECLARE
  nome TEXT;
BEGIN
  CASE mes
  WHEN 1 THEN nome = 'janeiro';
  WHEN 2 THEN nome = 'fevereiro';
  WHEN 3 THEN nome = 'março';
  WHEN 4 THEN nome = 'abril';
  WHEN 5 THEN nome = 'maio';
  WHEN 6 THEN nome = 'junho';
  WHEN 7 THEN nome = 'julho';
  WHEN 8 THEN nome = 'agosto';
  WHEN 9 THEN nome = 'setembro';
  WHEN 10 THEN nome = 'outubro';
  WHEN 11 THEN nome = 'novembro';
  WHEN 12 THEN nome = 'dezembro';
  ELSE nome = 'INVÁLIDO';
  END CASE;
  RETURN nome;
END;
$$ LANGUAGE PLPGSQL;

CREATE OR REPLACE FUNCTION mes_extenso(mes DATE) RETURNS TEXT AS $$
DECLARE
  m INT;
BEGIN
  -- Obtém o mês da data
  m := extract (MONTH FROM mes);
  -- Chama a outra função para retornar o resultado
  RETURN mes_extenso(m);
END;
$$ LANGUAGE PLPGSQL;

SELECT mes_extenso(current_date);
```

Exercício 3:

Cria uma função que receba uma data, obtenha o número de anos desta em relação a data atual e retorne:

NÃO VOTA: Se a idade não permite voto;

VOTO FACULTATIVO: Se o voto nesta esta idade for facultativo;

VOTO OBRIGATÓRIO: Se o voto nesta idade for obrigatório.

Solução:

```
CREATE OR REPLACE FUNCTION verifica_voto(data DATE) RETURNS TEXT AS $$
DECLARE
  idade INT;
  voto TEXT;
BEGIN
  -- A função age obtém a diferença entre duas datas (anos, meses,
 ↪  dias)
  -- Em seguida apenas os anos são extraídos
  idade := extract (YEAR FROM age(current_date, data));
  IF (idade < 16) THEN
    voto := 'Você ainda não pode votar';
  ELSIF (idade < 18) OR (idade > 70) THEN
    voto := 'Voto facultativo';
  ELSE
    voto := 'Voto obrigatório';
  END IF;
  RETURN voto;
END;
$$ LANGUAGE PLPGSQL;

SELECT verifica_voto('2000-02-15');
SELECT verifica_voto('1997-01-20');
SELECT verifica_voto('1940-04-07');
SELECT verifica_voto('1991-10-25');
```

Exercício 4:

Crie uma função que calcule o fatorial de um número e outra que calcule o máximo divisor comum entre dois números.

Solução:

```
CREATE OR REPLACE FUNCTION fatorial(n INT) RETURNS INT AS $$
DECLARE
  fat INT := 1;
BEGIN
  FOR i IN 2..n LOOP
    fat := fat * i;
  END LOOP;
  RETURN fat;
END;
$$ LANGUAGE PLPGSQL;

SELECT fatorial(5);
SELECT fatorial(10);

CREATE OR REPLACE FUNCTION mdc(n1 INT, n2 INT) RETURNS INT AS $$
DECLARE
  aux INT;
  resto INT;
BEGIN
  -- Garante que o maior número é n1
  IF (n2 > n1) THEN
    aux := n1;
    n1 := n2;
    n2 := aux;
  END IF;
  LOOP
    resto := mod (n1, n2);
    EXIT WHEN resto = 0;
    n1 := n2;
    n2 := resto;
  END LOOP;
  RETURN n2;
END;
$$ LANGUAGE PLPGSQL;

SELECT mdc(48,8);
SELECT mdc(640,560);
```

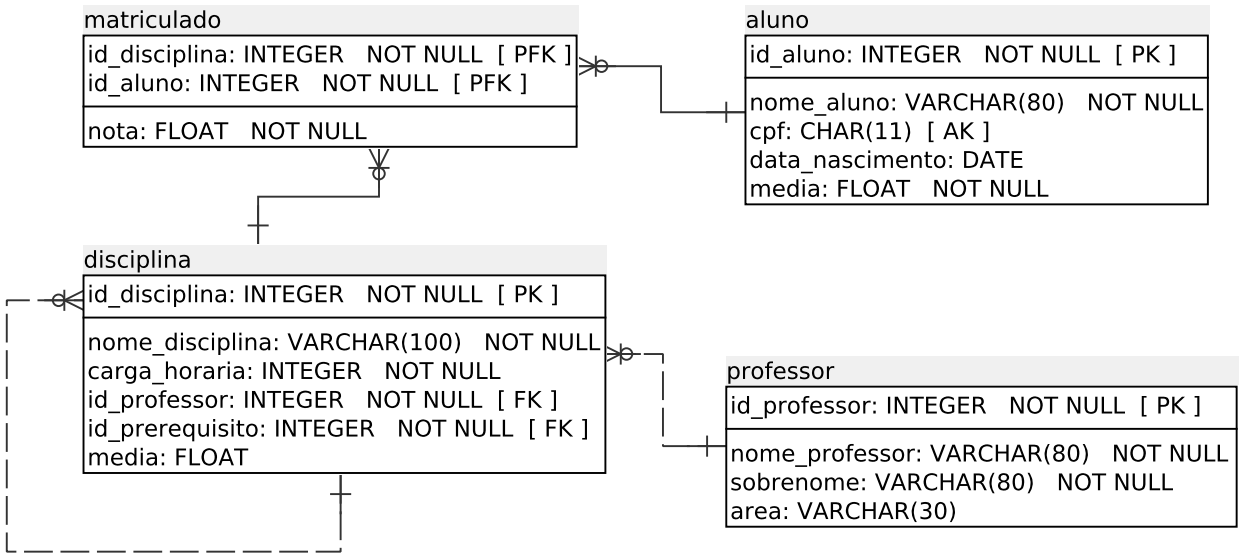

Figura 1: Banco de dados acadêmico II

Exercício 5:

Considere o banco de dados da Figura 1. Considere também que as notas iguais a **-1** (menos um) na tabela matriculado indicam que o aluno está apenas matriculado, mas não possui nenhuma nota. Escreva as instruções SQL para:

(a) Criar uma função com um laço para percorrer a tabela aluno e retornar id do aluno, nome do aluno e número de disciplinas que cada aluno for aprovado;

Solução:

```
CREATE OR REPLACE FUNCTION alunos_aprovacoes() RETURNS SETOF RECORD AS $$
DECLARE
  l RECORD;
BEGIN
  -- Laço que percorre todos os alunos
  FOR l IN SELECT id_aluno, nome_aluno FROM aluno LOOP
    -- Retorna o aluno atual e seu número de aprovações
    RETURN QUERY
      SELECT m.id_aluno, l.nome_aluno, COUNT(*) AS  aprovacoes
      FROM matriculado AS m
      WHERE m.id_aluno = l.id_aluno
        AND m.nota >= 60
      GROUP BY id_aluno;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE PLPGSQL;

SELECT * FROM alunos_aprovacoes()
AS (id INT, nome VARCHAR, aprovacoes BIGINT);
```

(b) Criar uma função que receba o id de uma disciplina e retorne os id dos alunos, nomes do aluno e notas, se a notas do alunos estiverem entre a média e a nota máxima da disciplina;

Solução:

```
CREATE OR REPLACE FUNCTION alunos_med_max(id_dis INT) RETURNS SETOF
 ↪  RECORD AS $$
DECLARE
  max REAL;
  med REAL;
  l RECORD;
BEGIN
  -- Obtém a nota máxima da disciplina recebida
  SELECT MAX(nota) INTO max FROM matriculado
  WHERE id_disciplina = id_dis;

  -- Obtém a média da disciplina recebida
  SELECT AVG(nota) INTO med FROM matriculado
  WHERE id_disciplina = id_dis AND nota <> -1;

  -- Laço percorrendo a nota de cada aluno em matriculado para a
 ↪  disciplina recebida
  FOR l IN SELECT a.id_aluno, a.nome_aluno, m.nota
           FROM matriculado AS m, aluno AS a
           WHERE a.id_aluno = m.id_aluno
           AND m.id_disciplina = id_dis
  LOOP
    -- Retorna se a nota do aluno estiver entre a média e a máxima
    IF l.nota BETWEEN med AND max THEN
      RETURN NEXT l;
    END IF;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE PLPGSQL;

SELECT * FROM alunos_med_max(100)
AS (id INT, nome VARCHAR, notas FLOAT);
```

(c) Criar uma função com um laço para percorrer a tabela matriculado e retornar o registros cuja nota esteja acima da media da disciplina;

Solução:

```
CREATE OR REPLACE FUNCTION notas_acima_media() RETURNS SETOF
 ↪  matriculado AS $$
DECLARE
  media REAL;
  l RECORD;
BEGIN
  -- Laço que percorre a tabela matriculado
  FOR l IN SELECT * FROM matriculado
  LOOP
    -- Obtém a média da disciplina do registro atual
    SELECT AVG(nota) INTO media FROM matriculado
    WHERE id_disciplina = l.id_disciplina
      AND nota <> -1;
    -- Retorna o registro se a nota for maior do que a média
    IF l.nota > media THEN
      RETURN NEXT l;
    END IF;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE PLPGSQL;

SELECT * FROM notas_acima_media();
```

(d) Criar um gatilho que atualize a média das disciplinas automaticamente (lembre-se de desconsiderar as matrículas sem notas);

Solução: (gatilho para cada linha)

```
CREATE OR REPLACE FUNCTION func_gat_atualiza_media_disciplina() RETURNS TRIGGER
 ↪  AS $$
DECLARE
  med REAL;
BEGIN
  -- Considera registros que deixaram de existir (OLD)
  IF TG_OP IN ('DELETE', 'UPDATE') THEN
    -- Obtém a média da disciplina
    SELECT AVG(nota) INTO med
    FROM matriculado
    WHERE id_disciplina = OLD.id_disciplina
      AND nota <> -1;

    -- Atualiza a  média da disciplina
    UPDATE disciplina AS d
    SET media = med
    WHERE id_disciplina = OLD.id_disciplina;
  END IF;

  -- Considera novos registros (NEW)
  IF TG_OP IN ('INSERT', 'UPDATE') THEN
    -- Obtém a média da disciplina
    SELECT AVG(nota) INTO med
    FROM matriculado
    WHERE id_disciplina = NEW.id_disciplina
      AND nota <> -1;
    -- Atualiza a  média da disciplina
    UPDATE disciplina AS d
    SET media = med
    WHERE id_disciplina = NEW.id_disciplina;
  END IF;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_media_disciplina
AFTER INSERT OR UPDATE OR DELETE
ON matriculado FOR EACH ROW
EXECUTE PROCEDURE func_gat_atualiza_media_disciplina();
```

Solução: **(gatilho para cada instrução)**

```
CREATE OR REPLACE FUNCTION func_gat_atualiza_media_disciplina() RETURNS TRIGGER
 ↪  AS $$
BEGIN
  -- Atualiza a média de todas as disciplinas
  UPDATE disciplina AS d
  SET media = med.media
  FROM (
    SELECT id_disciplina, AVG(nota) AS media
    FROM matriculado
    WHERE nota <> -1
    GROUP BY id_disciplina) AS med
  WHERE d.id_disciplina = med.id_disciplina;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_media_disciplina
AFTER INSERT OR UPDATE OR DELETE
ON matriculado FOR EACH STATEMENT
EXECUTE PROCEDURE func_gat_atualiza_media_disciplina();
```

(e) Criar um gatilho que matricule os novos alunos automaticamente nas disciplinas sem pré-requisitos;

Solução:

```
CREATE OR REPLACE FUNCTION func_gat_matricula_sem_pre() RETURNS
 ↪  TRIGGER AS $$
BEGIN
  INSERT INTO matriculado(id_aluno, id_disciplina, nota)
  SELECT NEW.id_aluno, d.id_disciplina, -1
  FROM disciplina AS d
  WHERE d.id_prerequisito IS NULL;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_matricula_sem_pre
AFTER INSERT ON aluno FOR EACH ROW
EXECUTE PROCEDURE func_gat_matricula_sem_pre();
```

(f) Criar um gatilho que matricule o aluno automaticamente nas disciplinas cujos pré-requisitos já foram cumpridos;

Solução:

```
CREATE OR REPLACE FUNCTION func_gat_matricula_pre_cumprido()
 ↪  RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO matriculado(id_aluno, id_disciplina, nota)
  SELECT m.id_aluno, d.id_disciplina, -1
  FROM matriculado AS m, disciplina AS d
  WHERE EXISTS (
    SELECT 1 FROM matriculado
    WHERE id_aluno = m.id_aluno
    AND id_disciplina = d.id_prerequisito
    AND nota >= 60);

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_matricula_pre_cumprido
AFTER INSERT OR UPDATE ON matriculado FOR EACH STATEMENT
EXECUTE PROCEDURE func_gat_matricula_pre_cumprido();
```

Exercício 6:

Considere o banco de dados de uma empresa de varejo cujo esquema lógico é apresentado na Figura 2. Escreva as instruções SQL para:

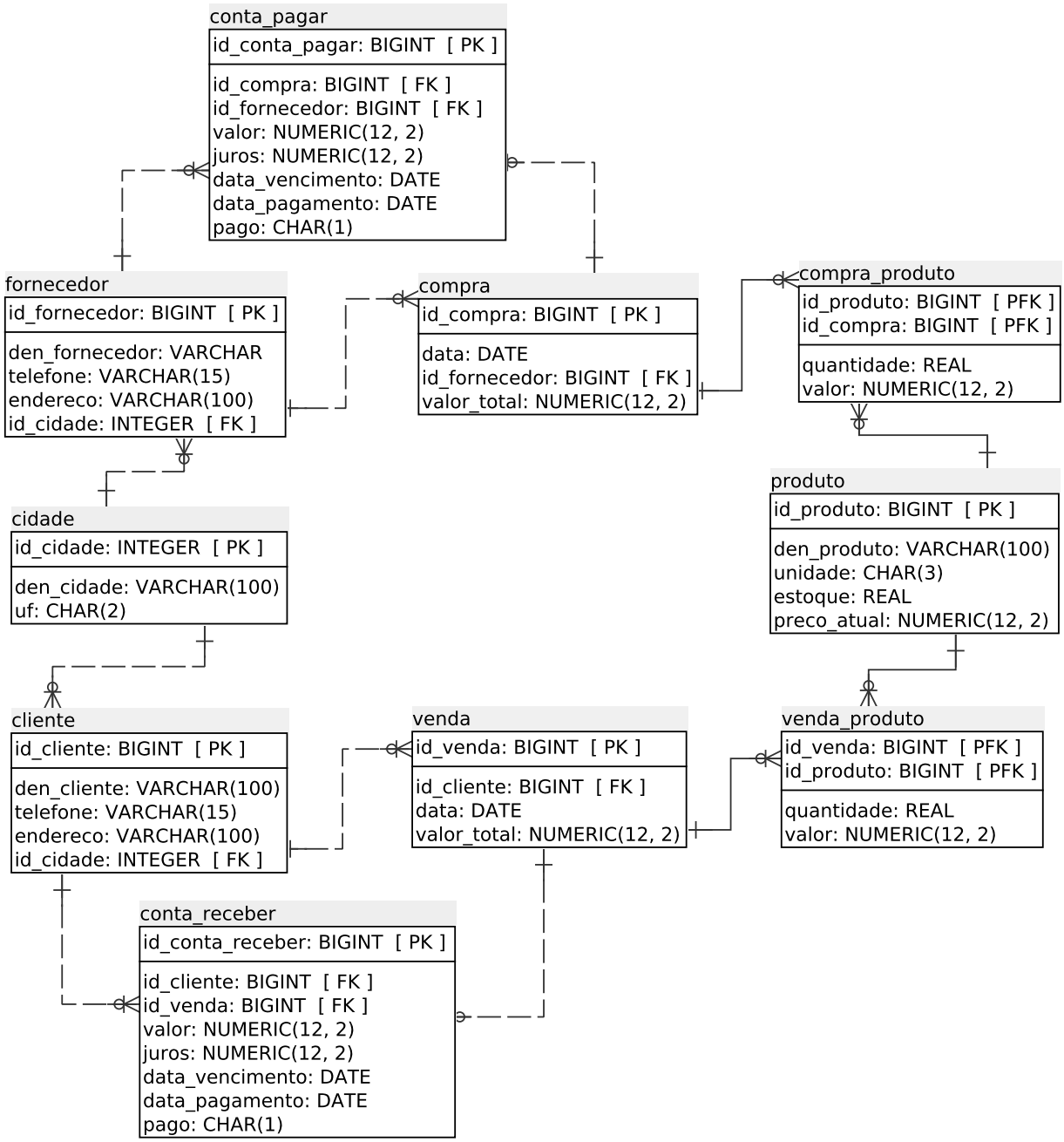

Figura 2: Banco de dados de uma empresa de varejo

(a) Criar uma função que receba o id de uma venda e atualize o valor total da mesma. Criar outra outra função análoga para compras. É possível fazer as duas coias só com uma função? Como?

Solução: (função para atualizar `venda`)

```
CREATE OR REPLACE FUNCTION atualiza_total_venda(id_ven INT) RETURNS VOID AS $$
DECLARE
  tot FLOAT;
BEGIN
  -- Calcula o valor total da venda
  SELECT SUM(quantidade * valor) AS valor_total
  INTO tot
  FROM venda_produto
  WHERE id_venda = id_ven;

  -- Atualiza o valor total
  UPDATE venda AS v
  SET valor_total = tot
  WHERE id_venda = id_ven;

  RETURN;
END;
$$ LANGUAGE PLPGSQL;

UPDATE venda SET valor_total = 0 WHERE id_venda = 10;
SELECT atualiza_total_venda(10);
SELECT * FROM venda WHERE id_venda = 10;
```

Solução: (função para atualizar `venda`)

```
CREATE OR REPLACE FUNCTION atualiza_total_compra(id_com INT) RETURNS VOID AS $$
DECLARE
  tot FLOAT;
BEGIN
  SELECT SUM(quantidade * valor) AS valor_total
  INTO tot
  FROM compra_produto
  WHERE id_compra = id_com;

  UPDATE compra AS v
  SET valor_total = tot
  WHERE id_compra = id_com;

  RETURN;
END;
$$ LANGUAGE PLPGSQL;

UPDATE compra SET valor_total = 0 WHERE id_compra = 10;
SELECT atualiza_total_compra(10);
SELECT * FROM compra WHERE id_compra = 10;
```

Solução: (função única)

```
CREATE OR REPLACE FUNCTION atualiza_total(id INT, tipo CHAR) RETURNS VOID AS $$
DECLARE
  tot FLOAT;
BEGIN
  -- Verifica se é compra ou venda
  IF (tipo = 'c') THEN
    SELECT SUM(quantidade * valor) AS valor_total
    INTO tot
    FROM compra_produto
    WHERE id_compra = id;

    UPDATE compra AS v
    SET valor_total = tot
    WHERE id_compra = id;
  ELSE
    SELECT SUM(quantidade * valor) AS valor_total
    INTO tot
    FROM venda_produto
    WHERE id_venda = id;

    UPDATE venda AS v
    SET valor_total = tot
    WHERE id_venda = id;
  END IF;

  RETURN;
END;
$$ LANGUAGE PLPGSQL;

UPDATE compra SET valor_total = 0 WHERE id_compra = 10;
SELECT atualiza_total(10, 'c');
SELECT * FROM compra WHERE id_compra = 10;

UPDATE venda SET valor_total = 0 WHERE id_venda = 10;
SELECT atualiza_total(10, 'v');
SELECT * FROM venda WHERE id_venda = 10;
```

(b) Criar um função com um laço percorrendo a tabela cliente que retorne id do cliente, nome do cliente e quantos produtos distintos cada cliente comprou.

Solução:

```
CREATE OR REPLACE FUNCTION cliente_num_produtos(id INT) RETURNS
 ↪  SETOF RECORD AS $$
DECLARE
  quant BIGINT;
  l RECORD;
BEGIN
  -- Laço que percorre a tabela de clientes
  FOR l IN SELECT id_cliente, den_cliente FROM cliente LOOP
    -- Calcula a quantidade de produtos
    WITH prod_dist AS (
      SELECT DISTINCT vp.id_produto
      FROM venda_produto AS vp, venda AS v
      WHERE v.id_venda = vp.id_venda
      AND v.id_cliente = l.id_cliente)
    SELECT COUNT(*) INTO quant FROM prod_dist;
    -- Retorna as informações do cliente atual
    RETURN QUERY SELECT l.id_cliente, l.den_cliente, quant;
  END LOOP;

  RETURN;
END;
$$ LANGUAGE PLPGSQL;

SELECT * FROM cliente_num_produtos(1)
AS (id BIGINT, nome VARCHAR, quant BIGINT);
```

(c) Criar gatilhos para atualizar automaticamente os estoques dos produtos de acordo com as compras e vendas;

Solução:

```
CREATE OR REPLACE FUNCTION func_gat_atualiza_estoque_compra() RETURNS TRIGGER
 ↪  AS $$
BEGIN
  -- Retira do estoque se uma compra for apagada
  IF TG_OP IN ('DELETE', 'UPDATE') THEN
    UPDATE produto
    SET estoque = estoque - OLD.quantidade
    WHERE id_produto = OLD.id_produto;
  END IF;

  -- Acrescenta uma nova compra no estoque
  IF TG_OP IN ('INSERT', 'UPDATE') THEN
    UPDATE produto
    SET estoque = estoque + NEW.quantidade
    WHERE id_produto = NEW.id_produto;
  END IF;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_estoque_compra
AFTER INSERT OR UPDATE OR DELETE
ON compra_produto FOR EACH ROW
EXECUTE PROCEDURE func_gat_atualiza_estoque_compra();

CREATE OR REPLACE FUNCTION func_gat_atualiza_estoque_venda() RETURNS TRIGGER AS
 ↪  $$
BEGIN
  IF TG_OP IN ('DELETE', 'UPDATE') THEN
    UPDATE produto
    SET estoque = estoque + OLD.quantidade
    WHERE id_produto = OLD.id_produto;
  END IF;

  IF TG_OP IN ('INSERT', 'UPDATE') THEN
    UPDATE produto
    SET estoque = estoque - NEW.quantidade
    WHERE id_produto = NEW.id_produto;
  END IF;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_estoque_venda
AFTER INSERT OR UPDATE OR DELETE
ON venda_produto FOR EACH ROW
EXECUTE PROCEDURE func_gat_atualiza_estoque_venda();
```

(d) Criar um gatilho que atualize automaticamente o total das vendas de acordo com os itens vendidos;

Solução:

```
CREATE OR REPLACE FUNCTION func_gat_atualiza_total_venda() RETURNS
 ↪  TRIGGER AS $$
DECLARE
  tot FLOAT;
BEGIN
  IF TG_OP IN ('DELETE', 'UPDATE') THEN
    SELECT SUM(quantidade * valor) AS valor_total
    INTO tot
    FROM venda_produto
    WHERE id_venda = OLD.id_venda;

    UPDATE venda AS v
    SET valor_total = tot
    WHERE id_venda = OLD.id_venda;
  END IF;

  IF TG_OP IN ('INSERT', 'UPDATE') THEN
    SELECT SUM(quantidade * valor) AS valor_total
    INTO tot
    FROM venda_produto
    WHERE id_venda = NEW.id_venda;

    UPDATE venda AS v
    SET valor_total = tot
    WHERE id_venda = NEW.id_venda;
  END IF;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_total_venda
AFTER INSERT OR UPDATE OR DELETE
ON venda_produto FOR EACH ROW
EXECUTE PROCEDURE func_gat_atualiza_total_venda();
```

(e) Criar um gatilho que atualize automaticamente o total das compras de acordo com os itens comprados;

Solução:

```
CREATE OR REPLACE FUNCTION func_gat_atualiza_total_compra() RETURNS
 ↪  TRIGGER AS $$
DECLARE
  tot FLOAT;
BEGIN
  IF TG_OP IN ('DELETE', 'UPDATE') THEN
    SELECT SUM(quantidade * valor) AS valor_total
    INTO tot
    FROM compra_produto
    WHERE id_compra = OLD.id_compra;

    UPDATE compra AS v
    SET valor_total = tot
    WHERE id_compra = OLD.id_compra;
  END IF;

  IF TG_OP IN ('INSERT', 'UPDATE') THEN
    SELECT SUM(quantidade * valor) AS valor_total
    INTO tot
    FROM compra_produto
    WHERE id_compra = NEW.id_compra;

    UPDATE compra AS v
    SET valor_total = tot
    WHERE id_compra = NEW.id_compra;
  END IF;

  RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_total_compra
AFTER INSERT OR UPDATE OR DELETE
ON compra_produto FOR EACH ROW
EXECUTE PROCEDURE func_gat_atualiza_total_compra();
```

(f) Crie os atributos de limite de crédito e saldo de crédito para os clientes. Atributa o valor de 30% do total de vendas de cada cliente para seu limite de crédito e para seu saldo de crédito. Criar um gatilho que atualize o saldo de crédito automaticamente de acordo com as vendas.

Solução:

```
ALTER TABLE cliente ADD limite_credito FLOAT;
ALTER TABLE cliente ADD saldo_credito FLOAT;

UPDATE cliente AS c
SET limite_credito = calc.limite, saldo_credito = calc.limite
FROM (
  SELECT id_cliente, SUM(valor_total) * 0.3 AS limite
  FROM venda
  GROUP BY id_cliente) AS calc
WHERE calc.id_cliente = c.id_cliente;

CREATE OR REPLACE FUNCTION func_gat_atualiza_saldo_cliente()
 ↪  RETURNS TRIGGER AS $$
DECLARE
  saldo FLOAT;
BEGIN
  SELECT saldo_credito INTO saldo
  FROM cliente WHERE id_cliente = NEW.id_cliente;

  IF (NEW.valor_total < saldo) THEN
    UPDATE cliente
    SET saldo_credito = saldo_credito - NEW.valor_total
    WHERE id_cliente = NEW.id_cliente;
    RETURN NEW;
  ELSE
    RAISE EXCEPTION 'Cliente % com saldo de crédito %
 ↪  insuficiente.', NEW.id_cliente, saldo;
    RETURN NULL;
  END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER gat_atualiza_saldo_cliente
BEFORE INSERT ON venda FOR EACH ROW
EXECUTE PROCEDURE func_gat_atualiza_saldo_cliente();
```