Exercício 1: Quais as vantagens dos procedimentos armazenados?

Simplifica as execuções de instruções SQL pela aplicação, transferência de parte da responsabilidade de processamento para o servidor e facilidade na manutenção, reduzindo a quantidade de alterações na aplicação.

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()

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;

SELECT mes\_extenso(3);

SELECT mes\_extenso(13);

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.

CREATE FUNCTION datas (data INT) RETURNS TEXT AS $$

DECLARE

permite TEXT;

BEGIN

CASE data

WHEN <16 THEN permite = ‘não pode‘;

WHEN >= 16 <=18 THEN permite = ‘facultativo’;

WHEN >=18 THEN permite = ‘deve’;

ELSE permite = 'INVÁLIDO';

END CASE;

RETURN permite;

END;

$$ LANGUAGE plpgsql;

SELECT datas (3);

SELECT datas (20);

Exercício 4: Crie uma função que calcule o fatorial de um número e outra que calcule o máximo divisor comum entre dois números.

CREATE OR REPLACE FUNCTION somatorio(n1 INT, n2 INT) RETURNS INT AS $$

DECLARE

soma INT := 0;

BEGIN

FOR i IN n1..n2 LOOP

soma := soma + i;

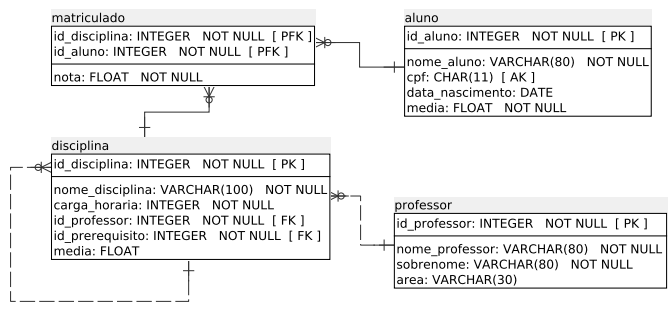
END LOOP;

RETURN soma;

END;

$$ LANGUAGE plpgsql;

SELECT somatorio(5);



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;

CREATE FUNCTION soma\_nota(INT, INT, FLOAT)

RETURNS void AS $$

UPDATE matriculado

SET nota = nota + $3

WHERE id\_disciplina = $1

AND id\_aluno = $2;

$$ LANGUAGE SQL;

(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;

DROP FUNCTION soma\_nota(INT, INT, FLOAT);

CREATE FUNCTION soma\_nota(INT, INT, FLOAT)

RETURNS float AS $$

UPDATE matriculado

SET nota = nota + $3

WHERE id\_disciplina = $1

AND id\_aluno = $2;

SELECT nota

FROM matriculado

WHERE id\_disciplina = $1

AND id\_aluno = $2;

$$ LANGUAGE SQL;

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

CREATE FUNCTION melhora\_media(aluno)

RETURNS FLOAT AS $$

SELECT CASE

WHEN $1.media < 60 THEN 60

ELSE $1.media

END AS media\_sonho;

$$ LANGUAGE SQL;

SELECT a.\*, melhora\_media(a.\*)

FROM aluno AS a;

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

CREATE FUNCTION novo\_aluno(VARCHAR)

RETURNS aluno AS $$

INSERT INTO aluno(nome\_aluno) VALUES ($1);

SELECT \* FROM aluno WHERE nome\_aluno = $1

$$ LANGUAGE SQL;

CREATE FUNCTION novo\_aluno(VARCHAR, CHAR)

RETURNS aluno AS $$

INSERT INTO aluno(nome\_aluno, cpf) VALUES ($1, $2);

SELECT \* FROM aluno WHERE nome\_aluno = $1

$$ LANGUAGE SQL;

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

CREATE FUNCTION novo\_aluno(VARCHAR)

RETURNS aluno AS $$

INSERT INTO aluno(nome\_aluno) VALUES ($1);

SELECT \* FROM aluno WHERE nome\_aluno = $1

$$ LANGUAGE SQL;

CREATE FUNCTION novo\_aluno(VARCHAR, CHAR)

RETURNS aluno AS $$

INSERT INTO aluno(nome\_aluno, cpf) VALUES ($1, $2);

SELECT \* FROM aluno WHERE nome\_aluno = $1

$$ LANGUAGE SQL;

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

DROP FUNCTION soma\_nota(INT, INT, FLOAT);

CREATE FUNCTION soma\_nota(INT, INT, FLOAT)

RETURNS float AS $$

UPDATE matriculado

SET nota = nota + $3

WHERE id\_disciplina = $1

AND id\_aluno = $2;

SELECT nota

FROM matriculado

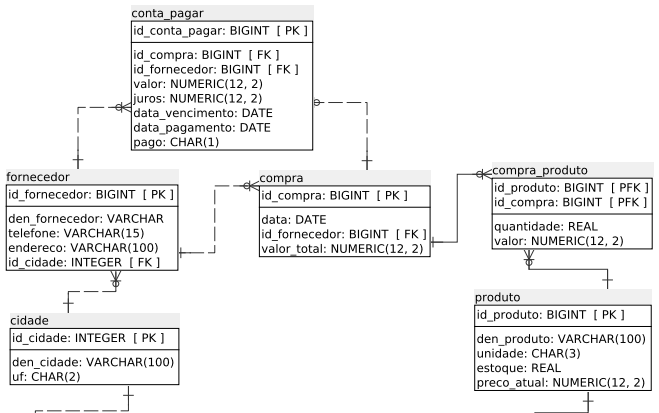
WHERE id\_disciplina = $1

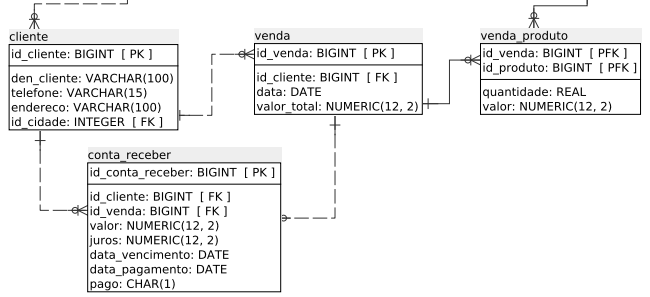
AND id\_aluno = $2;

$$ LANGUAGE SQL;

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:





(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?

CREATE FUNCTION atualiza\_venda()

RETURNS void AS $$

WITH vp AS (

SELECT id\_venda, AVG(venda) AS venda\_produto

FROM matriculado

WHERE venda <> -1

GROUP BY id\_venda)

UPDATE venda AS v

SET venda\_produto = venda

FROM vp

WHERE v.id\_venda = vp.id\_venda;

$$ LANGUAGE SQL;

(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.

CREATE FUNCTION atualiza\_venda()

RETURNS void AS $$

WITH id\_cliente AS (

SELECT id\_venda, AVG(venda) AS venda\_produto

FROM matriculado

WHERE venda <> -1

GROUP BY id\_venda)

UPDATE venda AS v

SET id\_cliente = cliente

FROM ic

WHERE v.id\_venda = ic.id\_venda;

$$ LANGUAGE SQL;

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

CREATE OR REPLACE VIEW estoque AS

SELECT id\_compra FROM compra;

CREATE TABLE compra2 (id\_compra INTEGER);

CREATE RULE \_RETURN AS

ON SELECT TO compra2 DO INSTEAD (

SELECT id\_compra FROM compra;

);

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

CREATE OR REPLACE VIEW estoque AS

SELECT vendas FROM compra;

CREATE TABLE compra2 (vendas INTEGER);

CREATE RULE \_RETURN AS

ON SELECT TO compra2 DO INSTEAD (

SELECT vendas FROM compra;

);

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

CREATE OR REPLACE VIEW total AS

SELECT id\_compra FROM compra;

CREATE TABLE ttotal (id\_compra INTEGER);

CREATE RULE \_RETURN AS

ON SELECT TO total DO INSTEAD (

SELECT id\_compra FROM 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.

ALTER TABLE cliente ADD limite\_credito FLOAT;

ALTER TABLE cliente ADD saldo\_credito FLOAT;

UPDATE INTO cliente

(id\_cliente, den\_cliente, telefone, endereço, id\_cidade, limite\_credito, saldo\_credito)

SELECT c.id\_cliente, c.den\_cliente, c.telefone, c.endereco, c.id\_cidade, 0.30 \* v.valor\_total, 0.30 \* v.valor\_total

FROM cliente AS c, venda AS v

WHERE c.id\_cliente = v.id\_cliente;