Trabalho de Laboratório de Banco de Dados

Apresento o

MarceDB – Sistema de Gestão para Marcenaria.

Aluno:

Matheus Henrique dos Santos Gomes

Professor(a):

Ivone Penque Matsuno Yugoshi

República Federativa do Brasil
Ministério da Educação
Fundação Universidade Federal do Mato Grosso do Sul

Sumário

1. INTRODUÇÃO **3**
2. DESCRIÇÃO DO CASO DE USO **3**
2.1 Requisitos para funcionamento do sistema: 3
3. ESQUEMA CONCEITUAL (DER) **5**
4. ESQUEMA LÓGICO (ESQUEMA RELACIONAL) **5**
5. SCRIPTS PARA CRIAÇÃO DO BANCO DE DADOS **6**
5.1 Tabela “cliente” 6
5.2 Tabela “modulo”: 6
5.3 Tabela “funcionario”: 7
5.4 Tabela “montador”: 7
5.5 Tabela “operador”: 8
5.6 Tabela “vendedor”: 8
5.7 Tabela “projeto”: 9
5.8 Tabela “estoqueChapa”: 10
5.9 Tabela “corte”: 11
5.10 Tabela “peca”: 12
5.11 Tabela “montaModulo”: 12
6. CONSULTAS **13**
6.1 Projetos de clientes de valor acima de 15 mil e local: 13
6.2 Cortes realizados no período de março de 2024: 13
6.3 Status dos projetos: 13
6.4 Desempenho do vendedor comparando salário com o total de vendas: 14
6.5 Quantidade de projetos por cliente: 14
6.6 Projetos que estão acima da média de valor: 14
6.7 Projetos dos clientes 1, 2, 4 e 8: 15
6.8 Projetos ainda não finalizados: 15
6.9 Clientes com projetos de alto padrão: 15
6.10 Cidades de atuação disponíveis: 15
6.11 Todos os detalhes de um projeto: 16
6.12 Projeto mais caro de cada cliente: 16
6.13 Projetos a partir de uma data: 16
6.14 Vendedores com mais de 50000 em vendas: 16
6.15 Total gasto e número de projetos de todos os clientes: 17
7. PROCEDIMENTOS **17**
7.1 Cálculo do bônus mensal dos funcionários: 17
7.2 Finalizar projeto e todos os seus cortes: 19
7.3 Gerar relatório mensal: 21
8. FUNÇÕES **22**
8.1 Ordem de compra para chapas que serão cortadas e não estão em estoque: 22
8.2 Verificar atividade do cliente nos últimos 12 meses: 23
8.3 Verificar se a garantia está expirada: 24
9. GATILHOS **25**
9.1 Atualizar complexidade do módulo automaticamente: 25
9.2 Validar data de venda do projeto: 26
9.3 Atualiza estoque após corte: 26
9.4 Coloca o projeto como finalizado ao fim de todos os cortes: 27
10. INFORMAÇÕES DOS ARQUIVOS **28**

1. INTRODUÇÃO

Em um cenário marcado pela crescente demanda por móveis planejados e a necessidade de otimização dos processos produtivos em marcenarias, surge a imperativa demanda por sistemas especializados que integrem e automatizam o ciclo completo de gestão. O MarceDB foi projetado como uma solução abrangente para marcenarias, unindo desde o primeiro contato comercial até a entrega final do produto, garantindo controle preciso sobre produção, estoque e relacionamento com clientes.

2. DESCRIÇÃO DO CASO DE USO

O sistema chamado MarceDB (Sistema de Gestão para Marcenaria) tem como objetivo registrar e acompanhar todo o ciclo de produção moveleira, desde o projeto inicial até a entrega final. O sistema tem como foco:

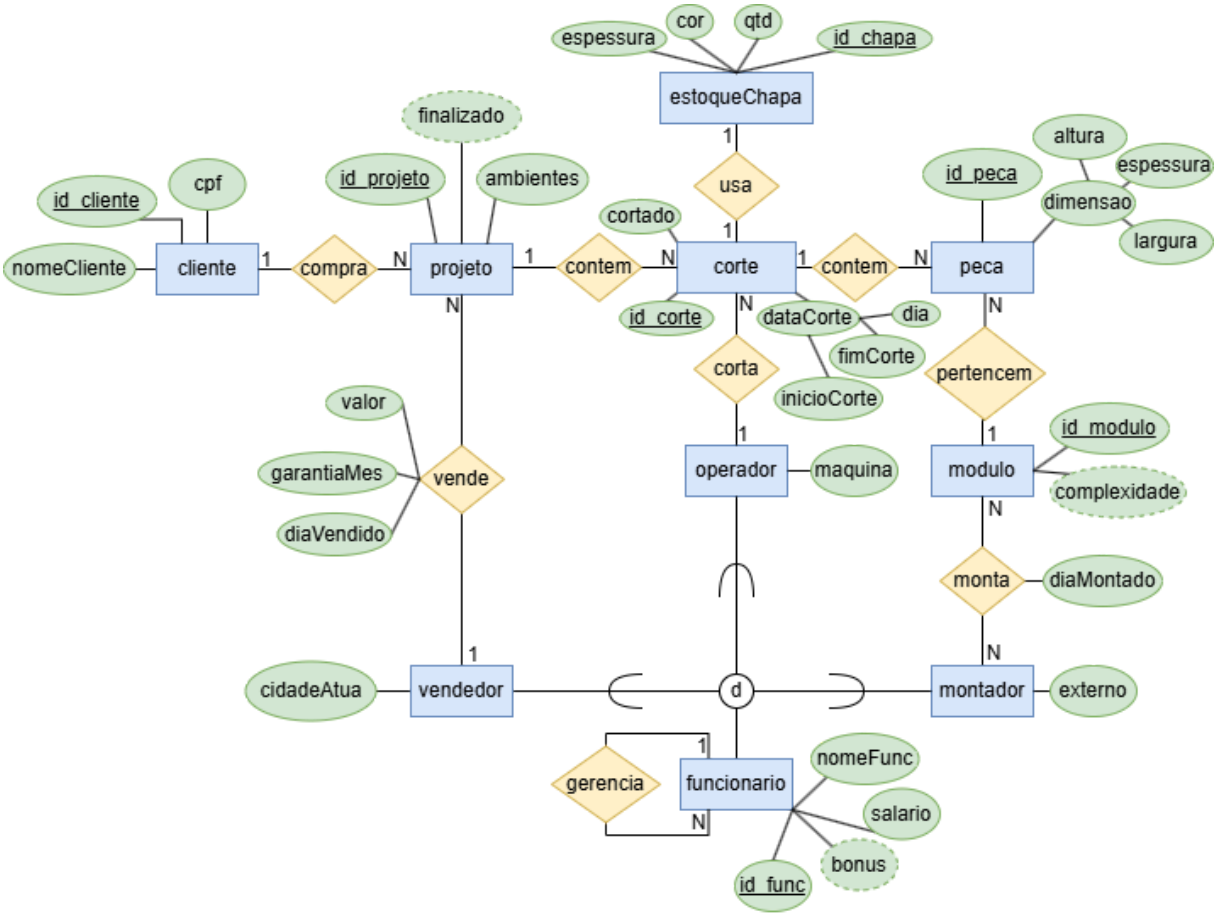
- O cadastro de participantes do processo (clientes, vendedores, operadores, montadores e gerentes),
- O controle de projetos(com valores, ambientes e prazos de garantia),
- O armazenamento de dados de produção(cortes, peças, módulos e estoque),
- O acompanhamento de produção por equipes especializadas e cálculo de bonificações.

2.1 Requisitos para funcionamento do sistema:

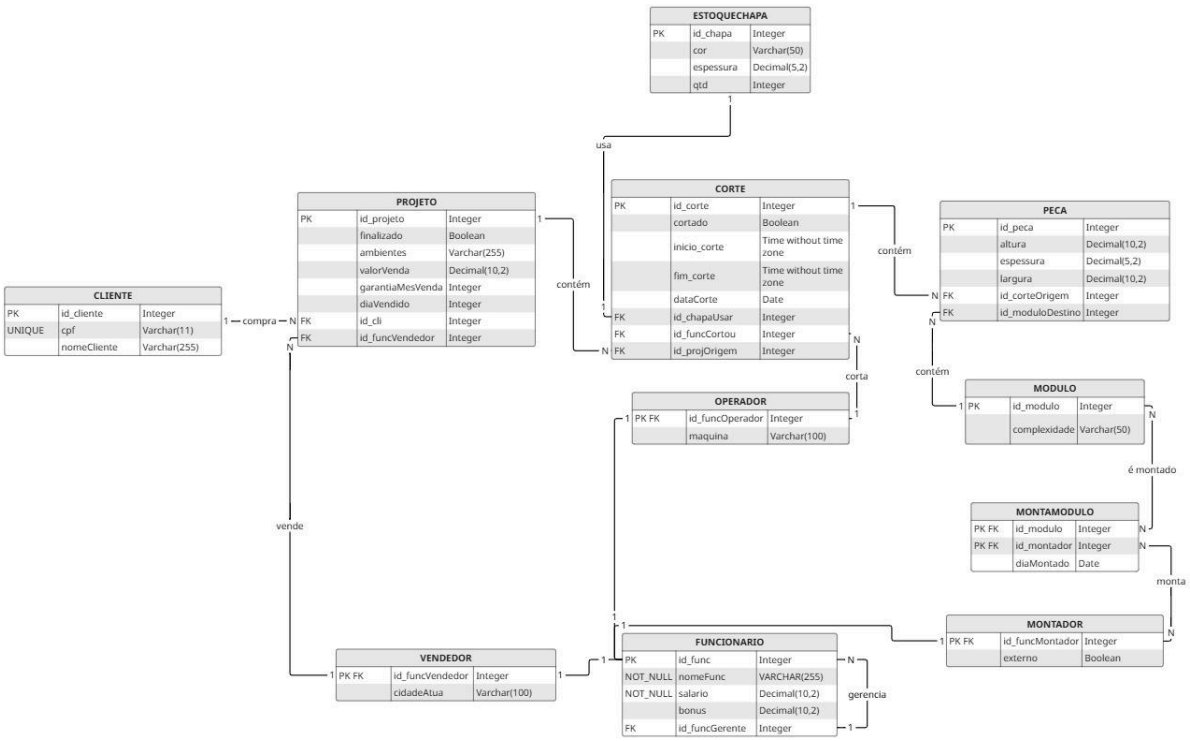
- Todo projeto registrado deve ter um número de identificação único, um status (em andamento ou finalizado), um cliente associado e uma descrição dos ambientes atendidos.
- O projeto pode ser de cliente cadastrado, associado a um CPF, ou avulso, com informações básicas de contato.
- Um cliente pode ter múltiplos projetos na marcenaria e ser classificado por tipo (Residencial, Comercial, Corporativo).
- Vendedores são identificados por ID único, cidade de atuação e podem acompanhar vários projetos simultaneamente.
- Projetos são processados através de cortes específicos, onde cada corte utiliza uma chapa do estoque e é executado por um operador qualificado.

- Cada corte deve registrar data de execução, horários de início e fim, e status de conclusão.
- Chapas de MDF são cadastradas com cor, espessura e quantidade em estoque, permitindo controle preciso do consumo.
- O sistema controla automaticamente a baixa no estoque quando cortes são finalizados.
- Projetos são compostos por módulos com níveis de complexidade calculados automaticamente baseado na quantidade de peças.
- Módulos são montados por equipe especializada (interna ou externa), com registro de data de montagem.
- Cada peça cortada possui dimensões precisas (altura, largura, espessura) em milímetros e está vinculada a um corte específico e módulo de destino.
- O sistema calcula automaticamente o aproveitamento de material e desperdícios.
- Funcionários possuem estrutura hierárquica com gerentes e subordinados, permitindo gestão por equipes.
- Sistema de bonificação diferenciado por função: vendedores (percentual sobre vendas), operadores (por corte executado), montadores (por complexidade de módulos).
- Projetos possuem garantia contratual em meses, com controle automático de vencimento através de funções especializadas.
- Histórico completo de alterações salariais é mantido para transparência na gestão de pessoas.
- O sistema gera relatórios mensais de produção com indicadores de eficiência, volume de vendas e desempenho por funcionário.
- Funções especializadas calculam ordem de compra, atividade de clientes e desperdício de material.
- *Triggers* garantem a integridade dos dados, validando datas de venda, níveis de garantia e disponibilidade de estoque.
- Impedem exclusões de projetos com cortes associados e mantém histórico de alterações críticas.

3. ESQUEMA CONCEITUAL (DER)



4. ESQUEMA LÓGICO (ESQUEMA RELACIONAL)



5. SCRIPTS PARA CRIAÇÃO DO BANCO DE DADOS

5.1 Tabela “cliente”

Tabela01_cliente				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_cliente	INTEGER	NOT NULL	PK(chave primária)	
cpf	VARCHAR(11)	UNIQUE NOT NULL		
nomeCliente	VARCHAR(255)	NOT NULL		

```sql
CREATE TABLE cliente (
    id_cliente INTEGER PRIMARY KEY,
    cpf VARCHAR(11) UNIQUE NOT NULL,
    nomeCliente VARCHAR(255) NOT NULL
);
```

5.2 Tabela “modulo”:

Tabela02_modulo				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_modulo	INTEGER	NOT NULL	PK(chave primária)	
complexidade	INTEGER	DEFAULT 1 CHECK(complexidade>0 AND complexidade<6)		

```sql
CREATE TABLE modulo (
    id_modulo INTEGER PRIMARY KEY,
    complexidade INTEGER DEFAULT 1 CHECK(complexidade>0 AND
complexidade<6)
);
```

5.3 Tabela “funcionario”:

Tabela03_funcionario				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_func	INTEGER	NOT NULL	PK(chave primária)	
nomeFunc	VARCHAR(255)	NOT NULL		
salario	INTEGER	NOT NULL		
bonus	DECIMAL(10,2)	DEFAULT 0		
id_funcGerente	INTEGER			FK(Chave estrangeira de funcionario)

```sql
CREATE TABLE funcionario (
    id_func INTEGER PRIMARY KEY,
    nomeFunc VARCHAR(255) NOT NULL,
    salario DECIMAL(10, 2) NOT NULL,
    bonus DECIMAL(10, 2) DEFAULT 0,
    id_funcGerente INTEGER,
    FOREIGN KEY (id_funcGerente) REFERENCES funcionario(id_func)
);
```

5.4 Tabela “montador”:

Tabela04_montador				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_funcMontador	INTEGER	NOT NULL	PK(chave primária)	FK(Chave estrangeira de funcionario)
externo	BOOLEAN			

```sql
CREATE TABLE montador (
    id_funcMontador INTEGER PRIMARY KEY,
    externo BOOLEAN,
    FOREIGN KEY (id_funcMontador) REFERENCES funcionario(id_func)
);
```

5.5 Tabela “operador”:

Tabela05_operador				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_funcOperador	INTEGER	NOT NULL	PK(chave primária)	FK(Chave estrangeira de funcionario)
maquina	VARCHAR(100)			

```sql
CREATE TABLE operador (
    id_funcOperador INTEGER PRIMARY KEY,
    maquina VARCHAR(100),
    FOREIGN KEY (id_funcOperador) REFERENCES funcionario(id_func)
);
```

5.6 Tabela “vendedor”:

Tabela06_vendedor				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_funcVendedor	INTEGER	NOT NULL	PK(chave primária)	FK(Chave estrangeira de funcionario)
cidadeAtua	VARCHAR(100)			

```sql
CREATE TABLE vendedor (
    id_funcVendedor INTEGER PRIMARY KEY,
    cidadeAtua VARCHAR(100),
    FOREIGN KEY (id_funcVendedor) REFERENCES funcionario(id_func)
);
```

5.7 Tabela “projeto”:

Tabela07_projeto				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_projeto	INTEGER	NOT NULL	PK(chave primária)	
finalizado	BOOLEAN	DEFAULT FALSE		
ambientes	VARCHAR(255)			
valorVenda	DECIMAL(10, 2)			
diaVendido	DATE			
garantiaMesVenda	INTEGER			
id_cli	INTEGER	NOT NULL		FK(chave estrangeira de cliente)
id_funcVendedor	INTEGER	NOT NULL		FK(chave estrangeira de vendedor)

```sql
CREATE TABLE projeto (
    id_projeto INTEGER PRIMARY KEY,
    finalizado BOOLEAN DEFAULT FALSE,
    ambientes VARCHAR(255),
    valorVenda DECIMAL(10, 2),
    diaVendido DATE,
    garantiaMesVenda INTEGER,
    id_cli INTEGER NOT NULL,
    id_funcVendedor INTEGER NOT NULL,
    FOREIGN KEY (id_cli) REFERENCES cliente(id_cliente),
    FOREIGN KEY (id_funcVendedor) REFERENCES
vendedor(id_funcVendedor)
);
```

5.8 Tabela “estoqueChapa”:

Tabela08_estoqueChapa				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_chapa	INTEGER	NOT NULL	PK(chave primária)	
cor	VARCHAR (50)			
espessura	DECIMAL(5, 2)			
qtd	INTEGER			

```sql
CREATE TABLE estoqueChapa (
    id_chapa INTEGER PRIMARY KEY,
    cor VARCHAR(50),
    espessura DECIMAL(5, 2),
    qtd INTEGER
);
```

5.9 Tabela “corte”:

Tabela09_corte				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_corte	INTEGER	NOT NULL	PK(chave primária)	
cortado	BOOLEAN			
diaCortado	DATE	NOT NULL		
inicio_corte	TIME WITHOUT TIME ZONE			
fim_corte	TIME WITHOUT TIME ZONE			
id_chapaUsar	INTEGER	NOT NULL		FK(chave estrangeira de estoqueChapa)
id_funcCortou	INTEGER			FK(chave estrangeira de operador)
id_projOrigem	INTEGER	NOT NULL		FK(chave estrangeira de projeto)

```sql
CREATE TABLE corte (
    id_corte INTEGER PRIMARY KEY,
    cortado BOOLEAN,
    diaCortado DATE,
    inicio_corte TIME WITHOUT TIME ZONE,
    fim_corte TIME WITHOUT TIME ZONE,
    id_chapaUsar INTEGER NOT NULL,
    id_funcCortou INTEGER,
    id_projOrigem INTEGER NOT NULL,
    FOREIGN KEY (id_chapaUsar) REFERENCES estoqueChapa(id_chapa),
    FOREIGN KEY (id_funcCortou) REFERENCES
operador(id_funcOperador),
    FOREIGN KEY (id_projOrigem) REFERENCES projeto(id_projeto)
);
```

5.10 Tabela “peca”:

Tabela10_peca				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_peca	INTEGER	NOT NULL	PK(chave primária)	
altura	DECIMAL(10, 2)			
espessura	DECIMAL(5, 2)			
largura	DECIMAL(10, 2)			
id_corteOrigem	INTEGER	NOT NULL		FK(chave estrangeira de corte)
id_moduloDestino	INTEGER	NOT NULL		FK(chave estrangeira de modulo)

```sql
CREATE TABLE peca (
    id_peca INTEGER PRIMARY KEY,
    altura DECIMAL(10, 2),
    espessura DECIMAL(5, 2),
    largura DECIMAL(10, 2),
    id_corteOrigem INTEGER NOT NULL,
    id_moduloDestino INTEGER NOT NULL,
    FOREIGN KEY (id_corteOrigem) REFERENCES corte(id_corte),
    FOREIGN KEY (id_moduloDestino) REFERENCES modulo(id_modulo)
);
```

5.11 Tabela “montaModulo”:

Tabela11_montaModulo				
Atributo	Tipo de Dados	Restrição Domínio	Restrição de Entidade	Restrição Referencial
id_modulo	INTEGER	NOT NULL	PK(chave primária)	FK(chave estrangeira de modulo)
id_montador	INTEGER	NOT NULL	PK(chave primária)	FK(chave estrangeira de montador)
diaMontado	DATE			

```SQL
CREATE TABLE montaModulo (
    id_modulo INTEGER,
    id_montador INTEGER,
    diaMontado DATE,
    PRIMARY KEY (id_modulo, id_montador),
    FOREIGN KEY (id_modulo) REFERENCES modulo(id_modulo),
    FOREIGN KEY (id_montador) REFERENCES montador(id_funcMontador)
);
```

6. CONSULTAS

6.1 Projetos de clientes de valor acima de 15 mil e local:

```SQL
SELECT p.id_projeto, c.nomeCliente, v.cidadeAtua, p.valorVenda
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
JOIN vendedor as v ON p.id_funcVendedor=v.id_funcVendedor
WHERE p.valorVenda>15000;
```

6.2 Cortes realizados no período de março de 2024:

```SQL
SELECT c.id_corte, p.ambientes, c.diaCortado, c.inicio_corte,
c.fim_corte
FROM corte as c
JOIN projeto as p ON c.id_projOrigem=p.id_projeto
WHERE c.diaCortado BETWEEN '2024-03-01' AND '2024-03-31'
AND c.cortado=TRUE;
```

6.3 Status dos projetos:

```SQL
SELECT p.id_projeto, 'Finalizado' as situação
FROM projeto as p
WHERE p.finalizado=TRUE
```

```sql
UNION
SELECT p.id_projeto, 'Não Finalizado' as situação
FROM projeto as p
WHERE p.finalizado=FALSE;
```

6.4 Desempenho do vendedor comparando salário com o total de vendas:

```sql
SELECT v.id_funcVendedor, f.salario, SUM(p.valorVenda) as
total_vendas
FROM projeto as p
JOIN vendedor as v ON p.id_funcVendedor=v.id_funcVendedor
JOIN funcionario as f ON v.id_funcVendedor=f.id_func
GROUP BY v.id_funcVendedor, f.salario
ORDER BY total_vendas DESC;
```

6.5 Quantidade de projetos por cliente:

```sql
SELECT c.nomeCliente, COUNT(p.id_projeto) as qtd_projetos
FROM cliente as c
LEFT JOIN projeto as p ON c.id_cliente=p.id_cli
GROUP BY c.id_cliente, c.nomeCliente
ORDER BY qtd_projetos DESC;
```

6.6 Projetos que estão acima da média de valor:

```sql
SELECT p.id_projeto, c.nomeCliente, p.valorVenda
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
WHERE p.valorVenda>(SELECT AVG(valorVenda) FROM projeto)
ORDER BY p.valorVenda DESC;
```

6.7 Projetos dos clientes 1, 2, 4 e 8:

```sql
SELECT p.id_projeto, c.nomeCliente, p.ambientes
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
WHERE c.id_cliente IN (1, 2, 4, 8)
ORDER BY c.nomeCliente;
```

6.8 Projetos ainda não finalizados:

```sql
SELECT p.id_projeto, c.nomeCliente, p.ambientes, p.valorVenda
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
WHERE p.finalizado=FALSE
ORDER BY p.diaVendido;
```

6.9 Clientes com projetos de alto padrão:

```sql
SELECT DISTINCT c.id_cliente, c.nomeCliente
FROM cliente as c
JOIN projeto as p ON c.id_cliente=p.id_cli
WHERE p.valorVenda>30000;
```

6.10 Cidades de atuação disponíveis:

```sql
SELECT DISTINCT cidadeAtua
FROM vendedor
ORDER BY cidadeAtua;
```

6.11 Todos os detalhes de um projeto:

```sql
SELECT p.id_projeto, c.nomeCliente as cliente, v.cidadeAtua as
cidade, p.ambientes, p.valorVenda as preço, p.garantiaMesVenda as
tempo_garantia
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
JOIN vendedor as v ON p.id_funcVendedor=v.id_funcVendedor;
```

6.12 Projeto mais caro de cada cliente:

```sql
SELECT p.id_projeto, c.nomeCliente, p.ambientes, p.valorVenda
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
WHERE p.valorVenda=(
    SELECT MAX(p2.valorVenda)
    FROM projeto as p2
    WHERE p2.id_cli=p.id_cli
);
```

6.13 Projetos a partir de uma data:

```sql
SELECT p.id_projeto, p.ambientes, c.nomeCliente, p.diaVendido,
p.valorVenda
FROM projeto as p
JOIN cliente as c ON p.id_cli=c.id_cliente
WHERE p.diaVendido>='2024-06-14'
ORDER BY p.diaVendido DESC;
```

6.14 Vendedores com mais de 50000 em vendas:

```sql
SELECT v.id_funcVendedor, SUM(p.valorVenda) as total_vendas
FROM projeto as p
```

```
JOIN vendedor as v ON p.id_funcVendedor=v.id_funcVendedor
GROUP BY v.id_funcVendedor
HAVING SUM(p.valorVenda)>50000
ORDER BY total_vendas DESC;
```

6.15 Total gasto e número de projetos de todos os clientes:

```
SQL
SELECT c.nomeCliente, COUNT(p.id_projeto) as qtd_projetos,
       COALESCE(SUM(p.valorVenda), 0) as total_gasto
FROM cliente as c
LEFT JOIN projeto as p ON c.id_cliente=p.id_cli
GROUP BY c.id_cliente, c.nomeCliente
ORDER BY total_gasto DESC;
```

7. PROCEDIMENTOS

7.1 Cálculo do bônus mensal dos funcionários:

```
SQL
CREATE OR REPLACE PROCEDURE calcular_bonus_mensal(p_mes INTEGER,
p_ano INTEGER) AS $$
DECLARE
    total_vendas DECIMAL(10,2);
    total_cortes INTEGER;
    bonus_vendedor DECIMAL(10,2);
    bonus_operador DECIMAL(10,2);
    bonus_montador DECIMAL(10,2);
    reg RECORD;
BEGIN
    --bônus de vendedores em 1% das vendas
    FOR reg IN
        SELECT v.id_funcVendedor, COALESCE(SUM(p.valorVenda), 0) as
total_mes
        FROM vendedor as v
        LEFT JOIN projeto as p ON v.id_funcVendedor=p.id_funcVendedor
            AND EXTRACT(MONTH FROM p.diaVendido)=p_mes
```

```
            AND EXTRACT(YEAR FROM p.diaVendido)=p_ano
        GROUP BY v.id_funcVendedor
    LOOP
        bonus_vendedor:=reg.total_mes*0.01;
        UPDATE funcionario
        SET bonus=bonus_vendedor
        WHERE id_func=reg.id_funcVendedor;
        RAISE NOTICE 'Vendedor %: bônus de R$ % calculado sobre
vendas de R$ % no mês % do ano %', reg.id_funcVendedor,
bonus_vendedor, reg.total_mes, p_mes, p_ano;
    END LOOP;

    --bônus para operadores por corte de R$5,00
    FOR reg IN
        SELECT o.id_funcOperador, COUNT(c.id_corte) as cortes_mes
        FROM operador as o
        LEFT JOIN corte as c ON o.id_funcOperador=c.id_funcCortou
            AND EXTRACT(MONTH FROM c.diaCortado)=p_mes
            AND EXTRACT(YEAR FROM c.diaCortado)=p_ano
            AND c.cortado=TRUE
        GROUP BY o.id_funcOperador
    LOOP
        bonus_operador:=reg.cortes_mes*5.00;
        UPDATE funcionario
        SET bonus=bonus_operador
        WHERE id_func=reg.id_funcOperador;
        RAISE NOTICE 'Operador %: bônus de R$ % calculado sobre %
cortes no mês % do ano %', reg.id_funcOperador, bonus_operador,
reg.cortes_mes, p_mes, p_ano;
    END LOOP;

    --bônus para montadores pela soma da complexidade de tudo montado
por eles
    FOR reg IN
        SELECT m.id_funcMontador, COALESCE(SUM(md.complexidade), 0)
as complexidadeMontados
        FROM montador as m
        LEFT JOIN montamodulo as mm ON
m.id_funcMontador=mm.id_montador
```

```
            AND EXTRACT(MONTH FROM mm.diaMontado)=p_mes
            AND EXTRACT(YEAR FROM mm.diaMontado)=p_ano
        LEFT JOIN modulo as md ON mm.id_modulo=md.id_modulo
        GROUP BY m.id_funcMontador
    LOOP
        bonus_montador:=reg.complexidadeMontados;
        UPDATE funcionario
        SET bonus=bonus_montador
        WHERE id_func=reg.id_funcMontador;
        RAISE NOTICE 'Montador %: bônus de R$ % calculado sobre a
soma de complexidade dos módulos montados no mês % do ano %',
reg.id_funcMontador, bonus_montador, p_mes, p_ano;
    END LOOP;

    --colocar bônus=0 para os funcionários que não tiveram bônus
calculado
    UPDATE funcionario
    SET bonus=0
    WHERE id_func NOT IN (
        SELECT id_funcVendedor FROM vendedor
        UNION
        SELECT id_funcOperador FROM operador
        UNION
        SELECT id_funcMontador FROM montador
    );

    RAISE NOTICE 'Cálculo de bônus para %/% concluído!', p_mes,
p_ano;
END;
$$ LANGUAGE 'plpgsql';
```

7.2 Finalizar projeto e todos os seus cortes:

```
SQL
CREATE OR REPLACE PROCEDURE finalizar_projeto(p_id_projeto INTEGER)
AS $$
DECLARE
```

```
    v_projeto_exist BOOLEAN;
BEGIN
    --Verificar se projeto existe
    v_projeto_exist:=FALSE;

    SELECT EXISTS(SELECT 1 FROM projeto WHERE
id_projeto=p_id_projeto)
    INTO v_projeto_exist;

    IF v_projeto_exist=FALSE THEN
        RAISE EXCEPTION 'Projeto % não encontrado', p_id_projeto;
    END IF;

    --Verificar se já está finalizado
    IF (SELECT finalizado FROM projeto WHERE id_projeto=p_id_projeto)
THEN
        RAISE NOTICE 'Projeto % já está finalizado.', p_id_projeto;
        RETURN;
    END IF;

    -- Finalizar todos os cortes pendentes
    UPDATE corte
    SET cortado=TRUE,
        diaCortado=CURRENT_DATE,
        inicio_corte=COALESCE(inicio_corte, '08:00:00'),
        fim_corte=COALESCE(fim_corte, '17:00:00')
    WHERE id_projOrigem=p_id_projeto AND cortado=FALSE;

    -- Marcar projeto como finalizado
    UPDATE projeto SET finalizado=TRUE WHERE id_projeto=p_id_projeto;

    RAISE NOTICE 'Projeto % finalizado! Os cortes pendentes foram
concluídos.', p_id_projeto;
END;
$$ LANGUAGE 'plpgsql';
```

7.3 Gerar relatório mensal:

```sql
CREATE OR REPLACE PROCEDURE relatorio_mensal(p_mes INTEGER, p_ano
INTEGER) AS $$
DECLARE
    reg RECORD;
    projetos_vendidos INTEGER;
    valor_total_vendas DECIMAL(10,2);
    total_cortes INTEGER;
    media_cortes_dia DECIMAL(10,2);
    media_complexidade DECIMAL(10,2);
BEGIN
    RAISE NOTICE '=== RELATÓRIO - %/% ===', p_mes, p_ano;
    RAISE NOTICE '';

    --Parte de vendas
    SELECT COUNT(*), COALESCE(SUM(valorVenda), 0)
    INTO projetos_vendidos, valor_total_vendas
    FROM projeto
    WHERE EXTRACT(MONTH FROM diaVendido)=p_mes
        AND EXTRACT(YEAR FROM diaVendido)=p_ano;

    RAISE NOTICE 'VENDAS:';
    RAISE NOTICE 'Projetos vendidos: %', projetos_vendidos;
    RAISE NOTICE 'Valor total de vendas: R$ %', valor_total_vendas;
    RAISE NOTICE '';

    --Parte de cortes
    SELECT COUNT(*)
    INTO total_cortes
    FROM corte
    WHERE EXTRACT(MONTH FROM diaCortado)=p_mes
        AND EXTRACT(YEAR FROM diaCortado)=p_ano
        AND cortado=TRUE;

    IF total_cortes>0 THEN
        media_cortes_dia:=ROUND(total_cortes/24.0, 2);
    ELSE
        media_cortes_dia:=0;
    END IF;
```

```
    RAISE NOTICE 'PRODUÇÃO:';
    RAISE NOTICE 'Total de cortes: %', total_cortes;
    RAISE NOTICE 'Média de cortes por dias úteis: %',
media_cortes_dia;
    RAISE NOTICE '';

    --Parte de montagem
    SELECT COALESCE(AVG(m.complexidade), 0)
    INTO media_complexidade
    FROM modulo as m
    JOIN montaModulo mm ON m.id_modulo=mm.id_modulo
    WHERE EXTRACT(MONTH FROM mm.diaMontado)=p_mes
        AND EXTRACT(YEAR FROM mm.diaMontado)=p_ano;

    RAISE NOTICE 'MÓDULOS:';
    RAISE NOTICE 'Média de complexidade dos módulos montados: %',
ROUND(media_complexidade, 2);
END;
$$ LANGUAGE 'plpgsql';
```

8. FUNÇÕES

8.1 Ordem de compra para chapas que serão cortadas e não estão em estoque:

```
SQL
CREATE OR REPLACE FUNCTION ordem_compra()
RETURNS SETOF RECORD AS $$
DECLARE
    reg RECORD;
BEGIN
    FOR reg IN
        SELECT
            ec.id_chapa,
            ec.cor,
            ec.espessura,
            ec.qtd AS qtd_estoque,
```

```
            cp.qtd_necessaria,
            GREATEST(cp.qtd_necessaria-ec.qtd, 0) AS qtd_comprar
        FROM estoqueChapa as ec
        JOIN(
            SELECT c.id_chapaUsar, COUNT(*) AS qtd_necessaria
            FROM corte as c
            WHERE c.cortado=FALSE
            GROUP BY c.id_chapaUsar) as cp ON
ec.id_chapa=cp.id_chapaUsar
        WHERE cp.qtd_necessaria>ec.qtd
        ORDER BY qtd_comprar DESC
    LOOP
        RETURN NEXT reg;
    END LOOP;

    RETURN;
END;
$$ LANGUAGE 'plpgsql';

--Como usar:
--SELECT * FROM ordem_compra() AS (
--    id_chapa INTEGER,
--    cor VARCHAR(50),
--   espessura DECIMAL(5,2),
--    qtd_estoque INTEGER,
--    qtd_necessaria INTEGER,
--    qtd_comprar INTEGER
--);
```

8.2 Verificar atividade do cliente nos últimos 12 meses:

```
SQL
CREATE OR REPLACE FUNCTION atividade_cliente(p_id_cliente INTEGER)
RETURNS INTEGER AS $$
DECLARE
    compras_recentes INTEGER;
BEGIN
```

```
    compras_recentes:=0;

    SELECT COUNT(*)
    INTO compras_recentes
    FROM projeto
    WHERE id_cli=p_id_cliente
        AND diaVendido>=CURRENT_DATE-INTERVAL '12 months';

    RETURN compras_recentes;
END;
$$ LANGUAGE 'plpgsql';

--Como usar:
--SELECT atividade_cliente(1)
```

8.3 Verificar se a garantia está expirada:

```
SQL
CREATE OR REPLACE FUNCTION garantia_vencida(p_id_projeto INTEGER)
RETURNS TEXT AS $$
DECLARE
    data_vencimento DATE;
BEGIN
    SELECT (diaVendido + (garantiaMesVenda || '
months')::INTERVAL)::DATE
    INTO data_vencimento
    FROM projeto
    WHERE id_projeto = p_id_projeto;

    IF data_vencimento IS NULL THEN
        RETURN 'Projeto não encontrado';
    END IF;

    IF CURRENT_DATE > data_vencimento THEN
        RETURN 'garantia expirada';
    ELSE
        RETURN 'ainda em garantia';
    END IF;
```

```
END;
$$ LANGUAGE 'plpgsql';

--Como usar:
--SELECT garantia_vencida(1)
```

9. GATILHOS

9.1 Atualizar complexidade do módulo automaticamente:

```
SQL
CREATE OR REPLACE FUNCTION atualizar_complexidade_modulo()
RETURNS TRIGGER AS $$
BEGIN
    --Atualiza a complexidade baseada na quantidade de peças no
módulo
    UPDATE modulo
    SET complexidade=(
        SELECT
            CASE
                WHEN COUNT(*) BETWEEN 1 AND 3 THEN 1
                WHEN COUNT(*) BETWEEN 4 AND 6 THEN 2
                WHEN COUNT(*) BETWEEN 7 AND 9 THEN 3
                WHEN COUNT(*) BETWEEN 10 AND 12 THEN 4
                WHEN COUNT(*) > 12 THEN 5
                ELSE 1
            END
        FROM peca
        WHERE id_moduloDestino=COALESCE(NEW.id_moduloDestino,
OLD.id_moduloDestino)
    )
    WHERE id_modulo=COALESCE(NEW.id_moduloDestino,
OLD.id_moduloDestino);

    RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER trig_atualizar_complexidade
```

```
    AFTER INSERT OR UPDATE OR DELETE ON peca
    FOR EACH ROW
    EXECUTE FUNCTION atualizar_complexidade_modulo();
```

9.2 Validar data de venda do projeto:

```
SQL
CREATE OR REPLACE FUNCTION validar_data_venda()
RETURNS TRIGGER AS $$
BEGIN
    -- Não permite data de venda futura
    IF NEW.diaVendido>CURRENT_DATE THEN
        RAISE EXCEPTION 'Data de venda não pode ser futura: %',
NEW.diaVendido;
    END IF;

    -- Garantia mínima de 3 meses
    IF NEW.garantiaMesVenda<3 THEN
        RAISE EXCEPTION 'Garantia mínima deve ser de 3 meses';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER trig_validar_data_venda
    BEFORE INSERT OR UPDATE ON projeto
    FOR EACH ROW
    EXECUTE FUNCTION validar_data_venda();
```

9.3 Atualiza estoque após corte:

```
SQL
CREATE OR REPLACE FUNCTION atualizar_estoque_apos_corte()
RETURNS TRIGGER AS $$
BEGIN
    -- Quando um corte é marcado como realizado, reduz o estoque
    IF NEW.cortado=TRUE AND (OLD.cortado=FALSE OR OLD.cortado IS
NULL) THEN
```

```
        UPDATE estoqueChapa
        SET qtd=qtd-1
        WHERE id_chapa=NEW.id_chapaUsar
        AND qtd>0;

        --estoque mínimo de 5
        IF (SELECT qtd FROM estoqueChapa WHERE id_chapa =
NEW.id_chapaUsar)<5 THEN
            RAISE NOTICE 'ALERTA: Estoque da chapa % está abaixo de 5
unidades', NEW.id_chapaUsar;
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER trig_atualizar_estoque
    AFTER UPDATE ON corte
    FOR EACH ROW
    EXECUTE FUNCTION atualizar_estoque_apos_corte();
```

9.4 Coloca o projeto como finalizado ao fim de todos os cortes:

```SQL
CREATE OR REPLACE FUNCTION verificar_finalizacao_projeto()
RETURNS TRIGGER AS $$
DECLARE
    total_cortes_projeto INTEGER;
    cortes_concluidos INTEGER;
    projeto_finalizado BOOLEAN;
BEGIN
    SELECT COUNT(*)
    INTO total_cortes_projeto
    FROM corte
    WHERE id_projOrigem = NEW.id_projOrigem;

    SELECT COUNT(*)
    INTO cortes_concluidos
```

```
    FROM corte
    WHERE id_projOrigem = NEW.id_projOrigem
    AND cortado = TRUE;

    projeto_finalizado:=(total_cortes_projeto=cortes_concluidos);

    UPDATE projeto
    SET finalizado=projeto_finalizado
    WHERE id_projeto=NEW.id_projOrigem;
    IF projeto_finalizado THEN
        RAISE NOTICE 'Projeto % FINALIZADO!', NEW.id_projOrigem;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
CREATE TRIGGER trig_verificar_finalizacao_projeto
    AFTER UPDATE OF cortado ON corte
    FOR EACH ROW
    EXECUTE FUNCTION verificar_finalizacao_projeto();
```

10. INFORMAÇÕES DOS ARQUIVOS

- O arquivo “tabelasMarceDB.sql” é para criação das tabelas do banco de dados para o MarceDB.
- O arquivo “insertsMarceDB.sql” é para o povoamento do banco de dados criado.
- O arquivo “selectsMarceDB.sql” contém exemplos de consultas no banco de dados da MarceDB.
- O arquivo “MarceDB – Sistema de Gestão para Marcenaria.pdf” é o relatório técnico com as informações necessárias para todo o entendimento do estudo de caso.
- O arquivo “DERMarceDB.png” é uma representação gráfica conhecida como Diagrama Entidade-Relacionamento.
- O arquivo “ERMarceDB.png” é uma representação gráfica conhecida como Esquema-relacional.

Todos os arquivos são de autoria própria e foram feitos nas seguintes ferramentas: “Visual Studio Code”, “Draw.io”, “Miro.com” “Google Docs” e “Google Sheets”. Sendo utilizado como ferramenta de testes a “pdAdmin4” do “PostgreSQL”.