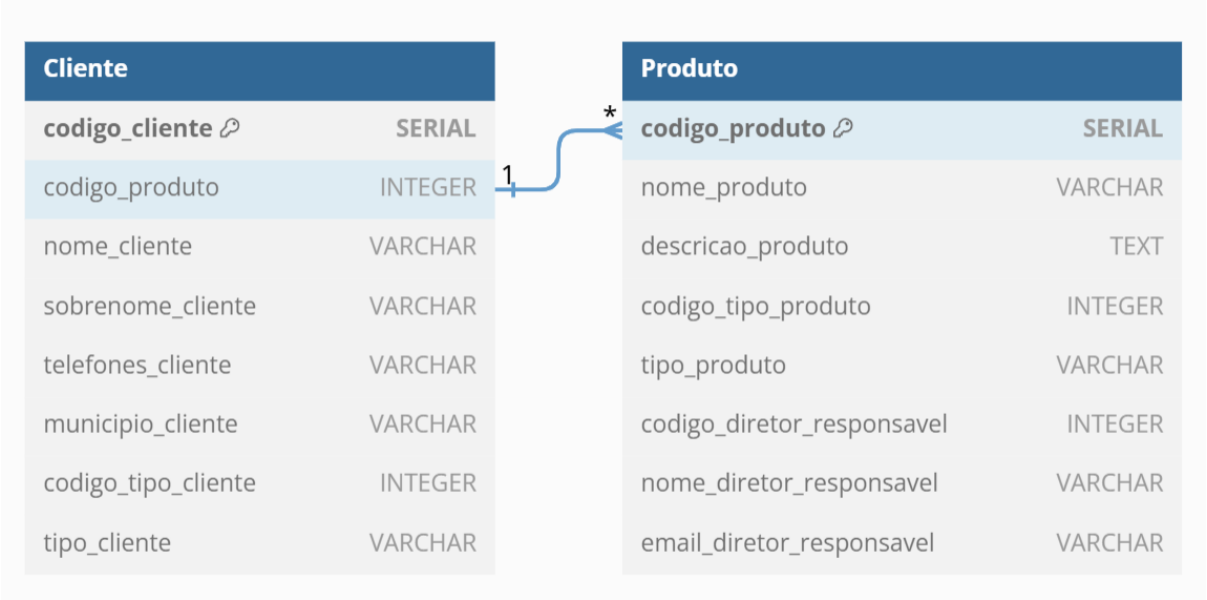
Avaliação Bancos de Dados I

Parte A - Modelagem e normalização de bancos de dados relacionais

1) Ainda sem fazer normalizações, apresente o modelo conceitual deste esboço oferecido pelo gestor, destacando atributos chaves e multivalorados, caso existam, e apresentando também a cardinalidade dos relacionamentos.



O modelo conceitual é a representação abstrata dos dados de um sistema de informação. O modelo solicitado na questão é representado por meio do diagrama e mostram os relacionamentos entre os dados.

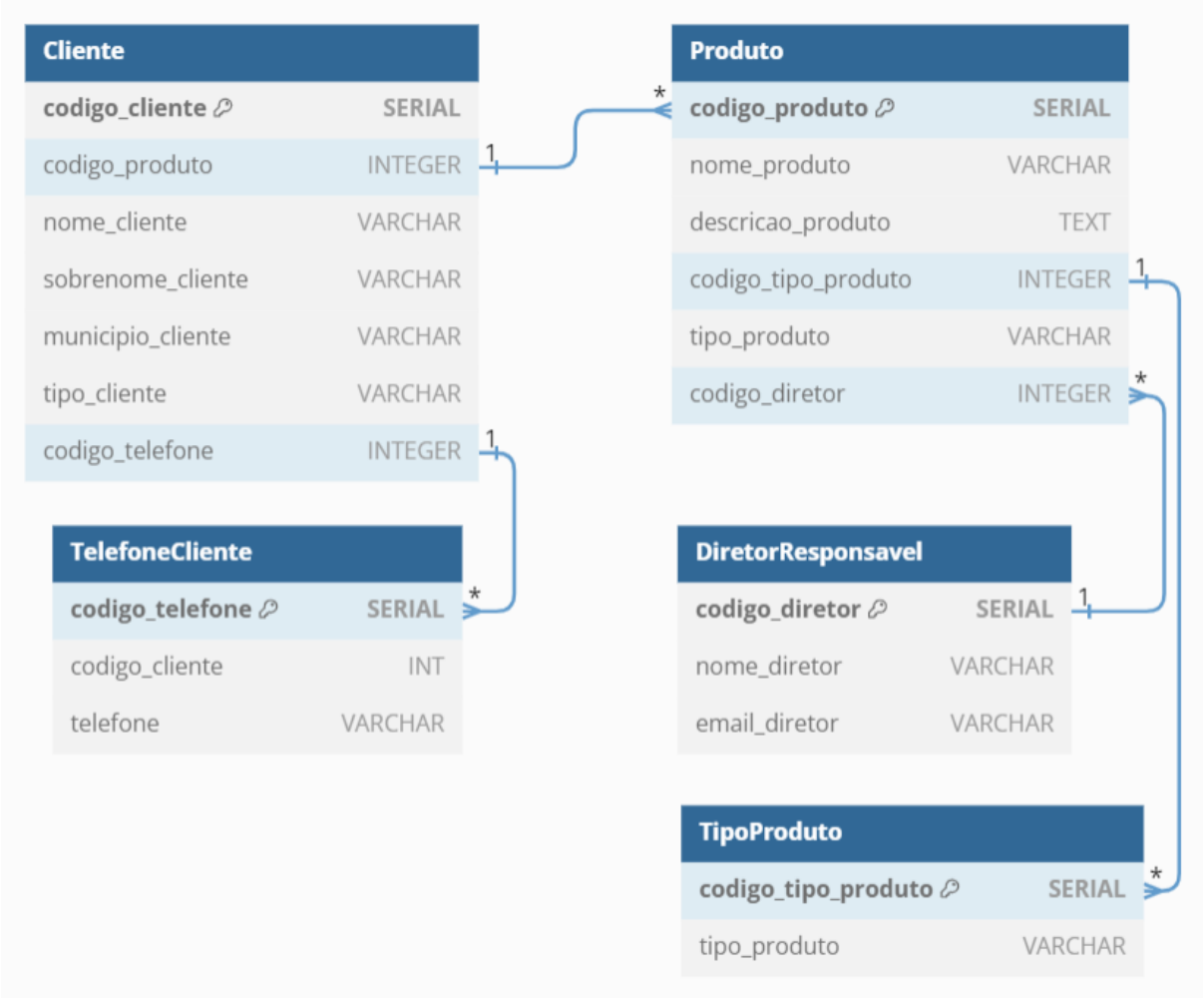
Os atributos chaves são aqueles que identificam de forma única cada registro em uma tabela. No nosso modelo, os atributos chaves são os seguintes: Cliente ➔ codigo_cliente e Produto ➔ codigo_produto.

Os atributos multivalorados são aqueles que podem conter vários valores para um mesmo registro. No nosso modelo, o atributo *telefones_cliente* da tabela Cliente pode ser multivalorado se um cliente puder ter mais de um telefone.

A cardinalidade dos relacionamentos indica o número de registros que podem estar relacionados a um único registro em outra tabela. No nosso modelo conceitual, a cardinalidade do relacionamento é 1 para N, ou seja, um cliente pode contratar vários produtos e um produto pode ser contratado por vários clientes.

Em um modelo de banco de dados relacional, quando há uma relação entre tabelas, geralmente é necessário adicionar chaves estrangeiras para garantir a integridade referencial. No caso apresentado não foi fornecido esse atributo, portanto criamos a chave estrangeira na tabela Cliente (*codigo_produto*).

2) Agora apresente um modelo lógico que expresse as mesmas informações e relacionamentos descritos no modelo original, mas decompondo-os quando necessário para que sejam respeitadas as 3 primeiras formas normais. Destaque atributos chaves e multivalorados, caso existam, e apresente também a cardinalidade dos relacionamentos.



Sobre os atributos chaves, multivalorados e a cardinalidade dos relacionamentos

Cliente
Atributo Chave: codigo_cliente
Cardinalidade:
Um cliente pode contratar vários produtos (1:N com a tabela Produto).

Um cliente pode ter vários telefones (1:N com a tabela TelefoneCliente).

TelefoneCliente
Atributos Chave: codigo_telefone
Cardinalidade: Um telefone pertence a um cliente (N:1 com a tabela Cliente).

Produto
Atributos Chave: codigo_produto
Cardinalidade:
Um produto pode ser contratado por vários clientes (N:1 com a tabela Cliente).
Um produto pode pertencer a vários tipos (N:1 com a tabela TipoProduto).
Um produto tem um diretor responsável (N:1 com a tabela DiretorResponsavel).

TipoProduto
Atributos Chave: codigo_tipo_produto
Cardinalidade: Vários produtos podem ter o mesmo tipo de produto (1:N com a tabela Produto).

DiretorResponsavel:
Atributos Chave: codigo_diretor
Cardinalidade: Vários produtos podem ter o mesmo diretor responsável (1:N com a tabela Produto).

Sobre as 3 Formas Normais

A 1ª Forma Normal é um estado de organização de dados em que as tabelas de um banco de dados não possuem valores repetidos ou grupos de valores em uma única célula. Cada campo contém um único valor atômico, garantindo a simplicidade e a consistência dos dados.
Na 2ª Forma Normal, além de atender aos requisitos da 1ª Forma Normal, a tabela não deve ter dependências parciais em relação à chave primária. Isso significa que cada atributo não chave deve depender completamente da chave primária, evitando redundâncias e mantendo a integridade dos dados.
A 3ª Forma Normal vai além, eliminando dependências transitivas. Isso significa que nenhum atributo não chave deve depender de outro atributo não chave. Cada atributo deve depender apenas da chave primária, promovendo uma estrutura de banco de dados mais eficiente, evitando redundâncias e melhorando a manutenção e a escalabilidade do sistema.

Parte B - Consultas SQL simples e complexas em um banco de dados postgres

Os scripts das respostas das questões a seguir encontram-se no arquivo scripts.sql.

3) Liste os nomes de todos os produtos que custam mais de 100 reais, ordenando-os primeiramente pelo preço e em segundo lugar pelo nome. Use alias para mostrar o nome da coluna nome como "Produto" e da coluna preco como "Valor". A resposta da consulta não deve mostrar outras colunas de dados.

Criamos as tabelas: produtos, categorias e produtos_categorias e fizemos inserção de dados nas mesmas.

```sql
SELECT
    nome AS "Produto",
    preco AS "Valor"
FROM
    produtos
WHERE
    preco > 100
ORDER BY
    preco, nome;
```

4) Liste todos os ids e preços de produtos cujo preço seja maior do que a média de todos os preços encontrados na tabela "produtos".

```sql
SELECT id, preco
FROM produtos
WHERE preco > (SELECT AVG(preco) FROM produtos)
```

5) Para cada categoria, mostre o preço médio do conjunto de produtos a ela associados. Caso uma categoria não tenha nenhum produto a ela associada,

esta categoria não deve aparecer no resultado final. A consulta deve estar ordenada pelos nomes das categorias.

```sql
SELECT
    c.nome       AS categoria,
    AVG(p.preco) AS preco_medio
FROM categorias c
LEFT JOIN produtos_categorias pc
    ON c.id = pc.categoria_id
LEFT JOIN produtos p
    ON pc.produto_id = p.id
GROUP BY c.id
ORDER BY c.nome;
```

Parte C - Inserções, alterações e remoções de objetos e dados em um banco de dados postgres

6) Com o objetivo de demonstrar o seu conhecimento através de um exemplo contextualizado com o dia-a-dia da escola, utilize os comandos do subgrupo de funções DDL para construir o banco de dados simples abaixo, que representa um relacionamento do tipo 1,n entre as entidades "aluno" e "turma".

```sql
CREATE TABLE aluno (
    id_aluno SERIAL PRIMARY KEY,
    nome_aluno VARCHAR(255) NOT NULL,
    aluno_alocado BOOLEAN,
    id_turma INT REFERENCES turma(id_turma)
);

CREATE TABLE turma (
    id_turma SERIAL PRIMARY KEY,
    codigo_turma VARCHAR(255) NOT NULL,
    nome_turma VARCHAR(255) NOT NULL
);
```

7) Agora que você demonstrou que consegue ser mais do que um simples usuário do banco de dados, mostre separadamente cada um dos códigos DML necessários para cumprir cada uma das etapas a seguir:

a) Inserir pelo menos duas turmas diferentes na tabela de turma;

```sql
INSERT INTO turma (codigo_turma, nome_turma) VALUES
    ('TURMA_A', 'Trilha Dados'),
    ('TURMA_B', 'Trilha Devops');
```

b) Inserir pelo menos 1 aluno alocado em cada uma destas turmas na tabela aluno (todos com NULL na coluna aluno_alocado);

```sql
INSERT INTO aluno (nome_aluno, aluno_alocado, id_turma) VALUES
    ('Carlos Franco', NULL, 1),
    ('Maria Silva', NULL, 2);
```

c) Inserir pelo menos 2 alunos não alocados em nenhuma turma na tabela aluno (todos com NULL na coluna aluno_alocado);

```sql
INSERT INTO aluno (nome_aluno, aluno_alocado, id_turma) VALUES
    ('Carlos Santos', NULL, NULL),
    ('Ana Pereira', NULL, NULL);
```

d) Atualizar a coluna aluno_alocado da tabela aluno, de modo que os alunos associados a uma disciplina recebam o valor True e alunos não associados a nenhuma disciplina recebam o valor False para esta coluna.

```sql
UPDATE aluno
SET aluno_alocado = CASE
    WHEN id_turma IS NOT NULL THEN TRUE
    ELSE FALSE
END;
```