1.1 - Comecei pela modelagem das tabelas e o que seria necessario para realizar a tarefa, separei em cinco tabelas, sendo elas;

- Tabela produto:

```
CREATE TABLE produto(
id serial PRIMARY KEY,
nome VARCHAR(50),
descricao TEXT,
preco DOUBLE PRECISION,
categoria_id INTEGER NOT NULL,
FOREIGN KEY (categoria_id) REFERENCES categoria(id)
);
```

Defini um identificador unico do produto, um nome usando VARCHAR para economia de memoria, descrição em text, preço em float e uma FOREIGN KEY para o id da tabela “categoria”;

- Tabela Cliente:

```
CREATE TABLE cliente(
id serial PRIMARY KEY,
nome VARCHAR(100),
email VARCHAR(100)
);
```

id pk, nome e email em VARCHAR;

- Tabela Categoria:

```
CREATE TABLE categoria(
id serial PRIMARY KEY,
nome VARCHAR(100)
);
```

id pk, nome em VARCHAR;

- Tabela produto_categoria:

```
CREATE TABLE produto_categoria(
produto_id INTEGER NOT NULL,
categoria_id INTEGER NOT NULL,
FOREIGN KEY (produto_id) REFERENCES produto(id),
FOREIGN KEY (categoria_id) REFERENCES categoria(id)
);
```

FOREIGN KEY tanto para o id da tabela produto quanto para a tabela categoria;

- Tabela Pedido:

```
CREATE TABLE pedido(
id serial PRIMARY KEY,
cliente_id INTEGER NOT NULL,
data DATE,
endereco_entrega TEXT)
```

id pk, cliente_id, data e um endereço de entrega em text

- Tabela item_pedido:

```
CREATE TABLE item_pedido(
id serial PRIMARY KEY,
pedido_id INTEGER NOT NULL,
produto_id INTEGER NOT NULL,
quantidade INTEGER,
FOREIGN KEY (pedido_id) REFERENCES pedido(id),
FOREIGN KEY (produto_id) REFERENCES produto(id)
);
```

id pk, quantidade e duas chaves estrangeiras, uma para o id do produto e outra para o de pedido.

1.2 - Com as tabelas feitas, encontrei dificuldade em popula-las, então consultei alguns materiais na internet e elaborei os seguintes scripts:

- Para a Categoria:

```
INSERT INTO categoria (nome)
SELECT 'Categoria' || seq AS nome FROM generate_series(1, 50) AS seq;
```

Gerei 50 categorias, no formato sequencial (Categoria1, Categoria2…Categoria50);

- Para Produto:

```
INSERT INTO produto (nome, descricao, preco, categoria_id)
SELECT 'Produto' || seq AS nome, 'Descrição do produto ' || seq AS descricao,
(random() * 1000)::numeric(10,2) AS preco, (random() * 49)::integer + 1 AS categoria_id FROM generate_series(1, 50) AS seq;
```

Gerei 50 entradas para, nome, descrição, preço e categoria_id;

- Para Cliente:

```
INSERT INTO cliente (nome, email)
SELECT 'Cliente' || seq AS nome, 'cliente' || seq || '@exemplo.com' AS email FROM generate_series(1, 50) AS seq;
```

Gerei 50 entradas sequenciais para cliente e para e-mail, com a devida formatação;

- Para pedido:

```
INSERT INTO pedido (data, endereco_entrega, cliente_id)
SELECT timestamp '2023-03-27 12:00:00' + ((random() * 7)::integer - 3) * interval '1 day' AS data,
 'Endereço de entrega ' || seq AS endereco_entrega,
 (random() * 50)::integer + 1 AS cliente_id FROM generate_series(1, 50) AS seq;
```

Gerei entradas para a data, endereço e id do cliente;

- Para item_pedido:

```
INSERT INTO item_pedido (quantidade, preco, produto_id, pedido_id)
SELECT (random() * 10)::integer + 1 AS quantidade,
produto.preco AS preco,
produto.id AS produto_id,
pedido.id AS pedido_id
FROM produto
CROSS JOIN pedido
WHERE random() < 0.5;
```

Gerei entradas aleatórias para quantidade, o valor da coluna preço é obtido pela tabela produto, CROSS JOIN combina todas as linhas da tabela produto com a tabela pedido, ou seja qualquer produto pode ser associado a qualquer pedido, com WHERE para limitar que cada combinação tenha 50% de chance de ser incluída na inserção

1.3 - Com as tabelas populadas, testei a integridade dos dados conforme proposto no desafio;

1. Todos produtos com nome, descrição e preço em ordem ASC de nome

```
SELECT nome, descricao, preco
FROM produto
ORDER BY Nome ASC;
```

2. Todas as categorias com nome e numero de produtos associados, em ordem ASC de categoria e nomes

```
SELECT categoria.nome, produto.nome, produto.id
FROM categoria
JOIN produto ON produto.categoria_id = categoria.id
WHERE produto.categoria_id IN (SELECT categoria.id FROM categoria)
ORDER BY categoria.nome ASC, produto.nome ASC
```

   a. Usando JOIN para categoria e produto com base no campo categoria_id na tabela produto, permitindo adicionar a categoria correspondente;
   b.
   c. Usando o WHERE para especificar a consulta com somente produtos que pertencem a pelo menos uma categoria, usando por fim o a subconsulta SELECT retorna a lista de todo ID de categoria;

3. Todos pedidos com data, endereço e a soma dos preços, em ordem DESC de data:

```
SELECT pedido.data, pedido.endereco_entrega, SUM(item_pedido.preco * item_pedido.quantidade) AS total_do_pedido
FROM Pedido
JOIN Item_Pedido ON Pedido.ID = Item_Pedido.Pedido_ID
GROUP BY Pedido.ID
ORDER BY Pedido.Data DESC;
```

   a. JOIN entre pedido e item_pedido com o campo de pedido_id, permitindo assim que itens relacionados a cada pedido sejam adicionados a consulta;

   b. Usando SUM para calcular o total do pedido;

4. Todos os produtos vendidos pelo menos uma vez, DESC de quantidade total vendida:

```
SELECT produto.nome, produto.descricao, produto.preco, SUM(item_pedido.quantidade) AS quantidade_total_vendida
FROM Produto
JOIN Item_Pedido ON Produto.ID = Item_Pedido.Produto_ID
GROUP BY Produto.ID
ORDER BY SUM(Item_Pedido.Quantidade) DESC;
```

   a. JOIN para combinar produto com item_pedido;
   b. agrupa pelo produto_id,

5. Todos os pedidos de um determinado cliente em um determinado periodo, em ordem ASC de data;

```
SELECT pedido.id, pedido.cliente_id, pedido.data, pedido.endereco_entrega
FROM pedido
JOIN item_pedido ON pedido.ID = item_pedido.pedido_ID
JOIN cliente ON pedido.cliente_ID = cliente.ID
WHERE cliente.nome IN (SELECT nome FROM cliente limit 1) AND pedido.data BETWEEN '2023-01-01' AND '2023-04-23'
ORDER BY cliente.nome ASC, pedido.data ASC;
```

   a. JOIN nas tabelas pedido e item_pedido, assim como nas tabelas pedido e cliente;
   b. WHERE para filtrar apenas um registro na tabela clientes, para retornar o cliente a ser analisado e determina o intervalo de tempo;

6. Listar produtos replicados e quantidade de replicações, ordem DESC de replicações:

```
SELECT produto.nome, COUNT(produto.nome) AS replicacoes
FROM produto
GROUP BY produto.nome
HAVING COUNT(produto.nome)>1
ORDER BY replicacoes DESC;
```

   a. COUNT para contar o numero de vezes que cada produto aparece na tabela;
   b. HAVING para filtrar apenas aqueles que aparecem mais de uma vez;

1.4 - Materiais consultados:

Como mencionado no inicio de 1.2, eis o material consultado para realizar a população das tabelas:
(https://dev.to/amrelmohamady/how-to-insert-random-data-for-testing-in-postgresql-2l98

https://dataschool.com/learn-sql/random-sequences/)