```sql
###Teste Técnico - Renato Druciak Regis

DROP TABLE IF EXISTS tb_customers;
CREATE TABLE tb_customers(
    customerId INT IDENTITY(1,1) PRIMARY KEY,
    customerDoc CHAR(14),
    firstName VARCHAR(32),
    lastName VARCHAR(32),
    birthDate DATE
);
-- Criação da tabela de produtos
DROP TABLE IF EXISTS tb_products;
CREATE TABLE tb_products(
    productId INT IDENTITY(1,1) PRIMARY KEY,
    productName VARCHAR(50),
    price DECIMAL(10, 2)
);

-- Criação da tabela de pedidos
DROP TABLE IF EXISTS tb_orders;
CREATE TABLE tb_orders(
    orderId INT IDENTITY(1,1) PRIMARY KEY,
    customerId INT FOREIGN KEY REFERENCES tb_customers(customerId),
    orderDate DATE
);

-- Criação da tabela de itens de pedidos
DROP TABLE IF EXISTS tb_order_items;
CREATE TABLE tb_order_items(
    orderItemId INT IDENTITY(1,1) PRIMARY KEY,
    orderId INT FOREIGN KEY REFERENCES tb_orders(orderId),
    productId INT FOREIGN KEY REFERENCES tb_products(productId),
    quantity INT
);

-- Índices para otimização de consultas
CREATE NONCLUSTERED INDEX ix_customerDoc    ON tb_customers(customerDoc) INCLUDE (customerId);
CREATE NONCLUSTERED INDEX ix_customerId     ON tb_orders(customerId);
CREATE NONCLUSTERED INDEX ix_orderId        ON tb_order_items(orderId);
CREATE NONCLUSTERED INDEX ix_productId      ON tb_order_items(productId);

--- População de dados na tabela de tb_customers
INSERT INTO tb_customers(customerDoc, firstName, lastName, birthDate)
SELECT
    RIGHT('00000000000' + CAST(ABS(CHECKSUM(NEWID())) % 100000000000000 AS VARCHAR), 11) AS customerDoc,
    LEFT(NEWID(), 32) AS firstName,
    LEFT(NEWID(), 32) AS lastName,
    DATEADD(DAY, ABS(CHECKSUM(NEWID())) % 36525, '1906-01-01') AS birthDate
FROM
    (SELECT TOP 1000 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS r FROM sys.columns) AS Numbers;

-- População de dados na tabela de produtos
INSERT INTO tb_products(productName, price)
VALUES
    ('Notebook', 2100.00),
    ('Smartphone', 1200.00),
    ('Tablet', 800.00),
    ('Headphones', 150.00),
    ('Monitor', 500.00),
    ('Keyboard', 80.00),
    ('Mouse', 40.00),
    ('Printer', 250.00),
    ('Webcam', 100.00),
    ('Speakers', 75.00),
    ('External Hard Drive', 130.00),
    ('USB Flash Drive', 20.00),
    ('Router', 90.00),
    ('Smartwatch', 250.00),
    ('Fitness Tracker', 100.00);

```

```sql
-- População de dados na tabela de pedidos
DECLARE @COUNT INT = 0, @LIM INT = 5
WHILE @COUNT <= @LIM BEGIN
    INSERT INTO tb_orders (customerId, orderDate)
    SELECT
        customerId,
        DATEADD(DAY, ABS(CHECKSUM(NEWID())) % 365, GETDATE()) AS orderDate
    FROM
        tb_customers
    WHERE
        customerId <= (SELECT ABS(CHECKSUM(NEWID())) % 1000)

    SET @COUNT += 1

END
;

-- População de dados na tabela de itens de pedidos
DECLARE @COUNT1 INT = 0, @LIM1 INT = 10
WHILE @COUNT1 <= @LIM1 BEGIN
INSERT INTO tb_order_items (orderId, productId, quantity)
SELECT
    o.orderId,
    p.productId,
    ABS(CHECKSUM(NEWID())) % 5 + 1 AS quantity
FROM
    tb_orders o
    CROSS JOIN tb_products p
WHERE
    ABS(CHECKSUM(NEWID())) % 10 < 3;

    SET @COUNT1 += 1
END
;


--################### RESPOSTAS ###################

--1.    Crie uma consulta que retorne apenas o item mais pedido e a quantidade total de pedidos.

select top 1
    tpr.productName as Produto
    ,count(tor.orderItemId) as Quantidade_de_pedidos
from tb_products tpr
left join tb_order_items tor on tor.productId = tpr.productId
group by tpr.productName
order by sum(tor.quantity) desc --quantidade vendida
;

--2.    Crie uma consulta que retorne todos os clientes que realizaram mais de 4 pedidos no último ano em ordem decrescente.
select
    tbc.customerId
    ,tbc.firstName
    ,tbc.lastName
    --,count(tor.orderId) as Numero_de_pedidos
from tb_customers tbc
left join tb_orders tor on tbc.customerId = tor.customerId
where tor.orderDate >= CONVERT(DATE, DATEADD(YEAR, -1, GETDATE()))
    --YEAR(tor.orderDate) = YEAR(GETDATE()) - 1 -- MESES DO ANO ANTERIOR
group by
    tbc.customerId
    ,tbc.firstName
    ,tbc.lastName
having count(tor.orderId) > 4
order by count(tor.orderId) desc
;

--3.    Crie uma consulta de quantos pedidos foram realizados em cada mês do último ano.
```

```sql
SELECT
    FORMAT(tor.orderDate, 'yyyy-MM') as Mes
    ,COUNT(tor.orderId) as Numero_de_pedidos
from tb_orders tor
where
    tor.orderDate BETWEEN DATEADD(DAY, 1, EOMONTH(GETDATE(), -13)) AND EOMONTH(GETDATE
    (), -1)
group by FORMAT(tor.orderDate, 'yyyy-MM')
order by FORMAT(tor.orderDate, 'yyyy-MM')
;



--4.    Crie uma consulta que retorne APENAS os campos "productName" e "totalAmount"
dos 5 produtos mais pedidos.

SELECT top 5
    tpr.productName,
    --sum(toi.quantity) as quantity,
    --tpr.price,
    sum(toi.quantity * tpr.price) as totalAmount
from tb_products tpr
left join tb_order_items toi on toi.productId = tpr.productId
GROUP BY tpr.productName
order by sum(toi.quantity) desc
;

--5.    Crie uma consulta liste todos os clientes que não realizaram nenhum pedido.

SELECT
    tbc.customerId
    ,tbc.firstName
    ,tbc.lastName
from tb_customers tbc
LEFT JOIN tb_orders tor on tor.customerId   = tbc.customerId
where
    tor.orderId is null
;


--6.    Crie uma consulta que retorne a data e o nome do produto do último
--pedido realizado pelos clientes onde o customerId são 94, 130, 300 e 1000.
SELECT DISTINCT
    --tor.customerId,
    tor.orderDate AS Data_ultimo_pedido
    ,tpr.productName AS Produto
FROM
    tb_orders               tor
INNER JOIN tb_order_items   toi ON tor.orderId      = toi.orderId
INNER JOIN tb_products      tpr ON toi.productId    = tpr.productId
WHERE
    tor.customerId IN (94, 130, 300, 1000)
AND tor.orderDate = (SELECT MAX(orderDate)
                     FROM tb_orders
                     WHERE customerId = tor.customerId)

;

--7.    Com base na estrutura das tabelas fornecidas (tb_order_items, tb_orders,
tb_products, tb_customers),
--crie uma nova tabela para armazenar informações sobre vendedores. A tabela deve
seguir os conceitos básicos
--de modelo relacional. Certifique-se de definir claramente as colunas da tabela e
suas relações
--com outras tabelas, se aplicável.

-- criação da tabela de vendedores
DROP TABLE IF EXISTS tb_sellers;
CREATE TABLE tb_sellers(
    sellerId INT IDENTITY(1,1) PRIMARY KEY,
    sellerDoc CHAR(14) NOT NULL,
    firstName VARCHAR(32) NOT NULL,
    lastName VARCHAR(32) NOT NULL,
```

```
209         commissionPerc DECIMAL(10, 4) NOT NULL,
210         CONSTRAINT UQ_sellerDoc UNIQUE (sellerDoc), -- Restringe sellerDoc para ser unico
211         CONSTRAINT CK_commissionPerc CHECK (commissionPerc >= 0 AND commissionPerc <= 0.50
            ) -- Restringe commissionPerc para ser entre 0 e 0.50
212     );
213
214     -- criação da tabela pedidos de vendedores, uma vez que a tabela tb_orders já existe
        no banco e desta forma podemos relacionar os vendedores com os pedidos
215     DROP TABLE IF EXISTS tb_order_seller;
216     CREATE TABLE tb_order_seller(
217         sellerOrderId INT IDENTITY(1,1) PRIMARY KEY,
218         sellerId INT FOREIGN KEY REFERENCES tb_sellers(sellerId),
219         orderId INT FOREIGN KEY REFERENCES tb_orders(orderId)
220     );
221
222     -- Índices para otimização de consultas
223     CREATE NONCLUSTERED INDEX ix_sellerDoc    ON tb_sellers(sellerDoc) INCLUDE (sellerId);
224     CREATE NONCLUSTERED INDEX ix_sellerId     ON tb_order_seller(sellerId);
225     CREATE NONCLUSTERED INDEX ix_orderId      ON tb_order_seller(orderId);
226
227     --- População de dados na tabela de tb_customers
228     INSERT INTO tb_sellers(sellerDoc, firstName, lastName, commissionPerc)
229     SELECT
230         RIGHT('00000000000' + CAST(ABS(CHECKSUM(NEWID())) % 100000000000000 AS VARCHAR),
            11) AS sellerDoc,
231         LEFT(NEWID(), 32) AS firstName,
232         LEFT(NEWID(), 32) AS lastName,
233         CAST((0.05 + (ABS(CHECKSUM(NEWID())) % 501) / 10000.0) AS DECIMAL(10,4)) AS
            commissionPerc
234     FROM
235         (SELECT TOP 5 ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS r FROM sys.columns) AS
             Numbers;
236
237     --8.    Crie uma procedure que insira dados na tabela de vendedores criada
        anteriormente.
238
239     --a.    Validar se o vendedor já existe na tabela.
240     --b.    Se o vendedor não existir, inserir um novo registro com os dados fornecidos.
241     --c.    Retornar uma mensagem indicando se a inserção foi bem-sucedida ou se o
        vendedor já está na tabela.
242     --      Escreva a implementação completa da procedure, incluindo a validação e a
        mensagem de retorno.
243
244     DROP PROCEDURE IF EXISTS usp_insert_seller;
245
246     CREATE PROCEDURE usp_insert_seller
247         @sellerDoc CHAR(14),
248         @firstName VARCHAR(32),
249         @lastName VARCHAR(32),
250         @commissionPerc DECIMAL(10, 4)
251     AS
252     BEGIN
253         SET NOCOUNT ON;
254
255         BEGIN TRY
256             DECLARE @message NVARCHAR(100);-- Variável para armazenar a mensagem de
                retorno
257
258             -- Verificar se o vendedor já existe
259             IF EXISTS (SELECT 1 FROM tb_sellers WHERE sellerDoc = @sellerDoc)
260             BEGIN
261                 SET @message = 'Vendedor já está na tabela.';-- Se o vendedor já existir,
                    envia a mensagem
262             END
263             ELSE
264             BEGIN
265                 -- Se o vendedor não existir, insere um novo registro
266                 INSERT INTO tb_sellers (sellerDoc, firstName, lastName, commissionPerc)
267                 VALUES (@sellerDoc, @firstName, @lastName, @commissionPerc);
268                 SET @message = 'Inserção bem-sucedida.'; -- Mensagem de retorno para
                    inserção bem-sucedida
269             END
270             SELECT @message AS Message;-- Retornar a mensagem
```

```
271         END TRY
272         BEGIN CATCH
273             SELECT ERROR_MESSAGE() AS ErrorMessage; --Capturar o erro e retornar a
                mensagem de erro
274         END CATCH
275     END;
276
277     --teste da procedure
278     EXEC usp_insert_seller @sellerDoc = '123123123', @firstName = 'asd', @lastName = 'qwe'
        , @commissionPerc = 7.50;
279
280     select * from tb_sellers;
281
282
283
284     --9.    Escreva um código em Python que se conecte a um banco de dados SQL Server e
        chame a procedure criada
285     --anteriormente para inserir um novo vendedor na tabela criada. Certifique-se de
        incluir o código de
286     --conexão ao banco de dados e a chamada da procedure com os parâmetros corretos.
287
288     #execução docker em wsl2 "sudo docker run -e 'ACCEPT_EULA=Y' -e
        'SA_PASSWORD=YourPassword!' -p 1433:1433 --name sqlserver -d
        mcr.microsoft.com/mssql/server:2019-latest"
289
290     import pymssql
291
292     # Função para inserir um novo vendedor
293
294     conection_string = {
295         'server': 'localhost',
296         'user': 'sa',
297         'password': 'YourPassword!',
298         'database': 'master'
299     }
300
301     vendedor = {
302         'sellerDoc': '123456789',
303         'firstName': 'João',
304         'lastName': 'Silva',
305         'commissionPerc': 0.05
306     }
307
308     def mysql_cursor(conection_string):
309         try:
310             # Conectar ao banco de dados
311             conn = pymssql.connect(**conection_string)
312             return conn.cursor(), conn
313         except pymssql.Error as e:
314             print(f"Erro ao conectar ao banco de dados: {e}")
315             return None, None
316
317
318     def insert_seller(cursor, conn, sellerDoc, firstName, lastName, commissionPerc):
319         try:
320             with cursor:
321                 cursor.callproc('usp_insert_seller', (sellerDoc, firstName, lastName,
                    commissionPerc))
322                 result = cursor.fetchall()
323                 for row in result:
324                     print(row)
325             conn.commit()
326         except Exception as e:
327             print("Erro ao executar a procedure:", e)
328             conn.rollback()
329         finally:
330             conn.close()
331
332
333     if __name__ == "__main__":
334         cursor, conn = mysql_cursor(conection_string)
335         if cursor and conn:
336             insert_seller(cursor, conn, **vendedor)
```

```
337
338
339 --10.   Em Python, crie um código que carregue em um “data frame” a tabela pedidos e
    a partir dele retorne
340 --os 10 produtos mais pedidos com as colunas "productName" e "numberOfOrders" em
    ordem decrescente.
341
342 import pymssql
343 import pandas as pd
344
345 conection_string = {
346     'server': 'localhost',
347     'user': 'sa',
348     'password': 'YourPassword!',
349     'database': 'master'
350 }
351
352 def mysql_cursor(conection_string):
353     try:
354         # Conectar ao banco de dados
355         conn = pymssql.connect(**conection_string)
356         return conn.cursor(), conn
357     except pymssql.Error as e:
358         print(f"Erro ao conectar ao banco de dados: {e}")
359         return None, None
360
361 def executa_query(cursor, query):
362     try:
363         cursor.execute(query)
364         return cursor.fetchall()
365     except pymssql.Error as e:
366         print(f"Erro ao executar a query: {e}")
367         return None
368
369
370 if __name__ == "__main__":
371     cursor, conn = mysql_cursor(conection_string)
372     if cursor and conn:
373         query_orders = "SELECT * FROM tb_order_items"
374         data_orders = executa_query(cursor, query_orders)
375         df_orders = pd.DataFrame(data_orders, columns=[desc[0] for desc in cursor.
            description])
376
377         query_products = "SELECT * FROM tb_products"
378         data_products = executa_query(cursor, query_products)
379         df_products = pd.DataFrame(data_products, columns=[desc[0] for desc in cursor.
            description])
380
381         # faz o join entre os dataframes df_orders e df_products
382         df = pd.merge(df_orders, df_products, left_on='productId', right_on=
            'productId', how='inner')
383         #faz os devidos agrupamentos
384         df = df.groupby('productName').agg({'orderItemId': 'nunique','quantity': 'sum'
            }).reset_index()
385         #altera o nome da coluna orderItemId para numberOfOrders, uma vez que a mesma
            representa a quantidade de pedidos daquele produto
386         df.rename(columns={'orderItemId': 'numberOfOrders'}, inplace=True)
387         #ordena a quantidade vendida em ordem decrescente
388         df = df.sort_values(by='quantity', ascending=False)
389         #manter apenas o top 10 em quantidade vendida no data frame
390         df = df[['productName', 'numberOfOrders']].head(10)
391         #reordena o dataframe de acordo com a coluna numberOfOrders em ordem
            decrescente
392         df = df.sort_values(by='numberOfOrders', ascending=False)
393         #retorna os 10 produtos mais pedidos com as colunas "productName" e
            "numberOfOrders" em ordem decrescente.
394         print(df)
395
396
397
398
399
```