Q.1. Create Tables For Sales Transaction Application
Note: Each table must have primary key, Unique key and should have foreign key constraints (can have check constraints)

```sql
--Product table

CREATE TABLE Product (
    ProductID INT PRIMARY KEY IDENTITY(1,1),
    ProductName NVARCHAR(255) NOT NULL,
    Description NVARCHAR(MAX),
    Price DECIMAL(10,2) NOT NULL CHECK (Price > 0),
    RemainingQuantity INT NOT NULL CHECK (RemainingQuantity >= 0)
);

--Customer table

CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY IDENTITY(1,1),
    Name NVARCHAR(255) NOT NULL,
    Email NVARCHAR(255) UNIQUE NOT NULL,
    Phone NVARCHAR(20) UNIQUE NOT NULL,
    CONSTRAINT CK_Customer_Phone CHECK (Phone LIKE '[0-9]%')
);

--Sale transactions table

CREATE TABLE SalesTransaction (
    TransactionID INT PRIMARY KEY IDENTITY(1,1),
    TransactionDate DATE NOT NULL,
    ProductID INT NOT NULL,
    CustomerID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    Total DECIMAL(10,2) NOT NULL CHECK (Total > 0),
    CONSTRAINT FK_SalesTransaction_Product FOREIGN KEY (ProductID) REFERENCES
Product(ProductID),
    CONSTRAINT FK_SalesTransaction_Customer FOREIGN KEY (CustomerID) REFERENCES
Customer(CustomerID)
);


--InvoiceTransaction Table

CREATE TABLE InvoiceTransaction (
    InvoiceID INT,
    TransactionID INT,
    PRIMARY KEY (InvoiceID, TransactionID),
    CONSTRAINT FK_InvoiceTransaction_Invoice FOREIGN KEY (InvoiceID) REFERENCES
Invoice(InvoiceID),
    CONSTRAINT FK_InvoiceTransaction_SalesTransaction FOREIGN KEY
(TransactionID) REFERENCES SalesTransaction(TransactionID)
);


--Invoice table

CREATE TABLE Invoice (
    InvoiceID INT PRIMARY KEY IDENTITY(1,1),
    InvoiceDate DATE NOT NULL,
    TotalBillAmount DECIMAL(10,2) NOT NULL CHECK (TotalBillAmount > 0)
);
```

Based on created tables, create the following using TSQL
Q.2. Store Procedures (Json as parameter for all SP preferred):

i. Create stored procedure for all CRUD operations.
ii. Create Invoice with following conditions:
• Once the invoice is generated for the customer, tag the sales transaction with the
correct invoice.
• Even if a customer buys multiple items, create a single invoice (bill) only.
• Should calculate a discount of 5% if the total invoice amount is less than or equal
to 1000 and 10% if the total invoice amount is greater than 1000.

CRUD Operations for PRODUCT:

```
--Create Stored Procedure for product


CREATE PROCEDURE sp_CreateProduct
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        INSERT INTO Product(ProductName, Description, Price, RemainingQuantity)
        SELECT ProductName, Description, Price, RemainingQuantity
        FROM OPENJSON(@json)
        WITH (
            ProductName NVARCHAR(255),
            Description NVARCHAR(MAX),
            Price DECIMAL(10,2),
            RemainingQuantity INT
        );
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT;
        SELECT @ErrMsg = ERROR_MESSAGE(),
               @ErrSeverity = ERROR_SEVERITY();
        RAISERROR(@ErrMsg, @ErrSeverity, 1);
    END CATCH;
END;


--Read Stored Procedure for Product


CREATE PROCEDURE sp_ReadProduct
    @json NVARCHAR(MAX)
AS
BEGIN
    DECLARE @ProductID INT;

    SELECT @ProductID = ProductID
    FROM OPENJSON(@json)
    WITH (
        ProductID INT
    );

    SELECT * FROM Product WHERE ProductID = @ProductID;
END;
```

```
--Update Stored Procedure for Product


CREATE PROCEDURE sp_UpdateProduct
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        DECLARE @ProductID INT, @ProductName NVARCHAR(255), @Description
NVARCHAR(MAX), @Price DECIMAL(10,2),@RemainingQuantity INT;

        SELECT @ProductID = ProductID, @ProductName = ProductName, @Description
= Description, @Price = Price, @RemainingQuantity=RemainingQuantity
        FROM OPENJSON(@json)
        WITH (
            ProductID INT,
            ProductName NVARCHAR(255),
            Description NVARCHAR(MAX),
            Price DECIMAL(10,2),
          RemainingQuantity INT
        );

        UPDATE Product
        SET ProductName = @ProductName, Description = @Description, Price =
@Price, RemainingQuantity=@RemainingQuanity
        WHERE ProductID = @ProductID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT;
        SELECT @ErrMsg = ERROR_MESSAGE(),
               @ErrSeverity = ERROR_SEVERITY();
        RAISERROR(@ErrMsg, @ErrSeverity, 1);
    END CATCH;
END;


--Delete Stored procedure for Product


CREATE PROCEDURE sp_DeleteProduct
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        DECLARE @ProductID INT;

        SELECT @ProductID = ProductID
        FROM OPENJSON(@json)
        WITH (
            ProductID INT
        );

        DELETE FROM Product WHERE ProductID = @ProductID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
```

```
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT;
        SELECT @ErrMsg = ERROR_MESSAGE(),
               @ErrSeverity = ERROR_SEVERITY();
        RAISERROR(@ErrMsg, @ErrSeverity, 1);
    END CATCH;
END;
```

CRUD Operations for CUSTOMER:

```
--Create Stored Procedure for Customer


CREATE PROCEDURE sp_CreateCustomer
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        INSERT INTO Customer(Name, Email, Phone)
        SELECT Name, Email, Phone
        FROM OPENJSON(@json)
        WITH (
            Name NVARCHAR(255),
            Email NVARCHAR(255),
            Phone NVARCHAR(20)
        );
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT;
        SELECT @ErrMsg = ERROR_MESSAGE(),
               @ErrSeverity = ERROR_SEVERITY();
        RAISERROR(@ErrMsg, @ErrSeverity, 1);
    END CATCH;
END;



--Read Stored Procedure for Customer


CREATE PROCEDURE sp_ReadCustomer
    @json NVARCHAR(MAX)
AS
BEGIN
    SELECT *
    FROM Customer
    WHERE CustomerID = (SELECT CustomerID FROM OPENJSON(@json) WITH (CustomerID
INT));
END;



--Update Stored Procedure for Customer


CREATE PROCEDURE sp_UpdateCustomer
    @json NVARCHAR(MAX)
AS
```

```
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE Customer
        SET Name = json.Name,
            Email = json.Email,
            Phone = json.Phone
        FROM OPENJSON(@json)
        WITH (
            CustomerID INT,
            Name NVARCHAR(255),
            Email NVARCHAR(255),
            Phone NVARCHAR(20)
        ) AS json
        WHERE Customer.CustomerID = json.CustomerID;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT;
        SELECT @ErrMsg = ERROR_MESSAGE(),
               @ErrSeverity = ERROR_SEVERITY();
        RAISERROR(@ErrMsg, @ErrSeverity, 1);
    END CATCH;
END;



--Delete Stored Procedure for Customer


CREATE PROCEDURE sp_DeleteCustomer
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        DELETE FROM Customer
        WHERE CustomerID = (SELECT CustomerID FROM OPENJSON(@json) WITH
(CustomerID INT));
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000), @ErrSeverity INT;
        SELECT @ErrMsg = ERROR_MESSAGE(),
               @ErrSeverity = ERROR_SEVERITY();
        RAISERROR(@ErrMsg, @ErrSeverity, 1);
    END CATCH;
END;
```

CRUD Operations for SALESTRANSACTION:

```
--Create Stored Procedure for salestransaction


CREATE PROCEDURE sp_CreateSalesTransaction
    @json NVARCHAR(MAX)
AS
```

```
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @TransactionDate DATE;
        DECLARE @ProductID INT;
        DECLARE @CustomerID INT;
        DECLARE @Quantity INT;
        DECLARE @Price DECIMAL(10,2);

        SELECT @TransactionDate = JSON_VALUE(@json, '$.TransactionDate'),
               @ProductID = JSON_VALUE(@json, '$.ProductID'),
               @CustomerID = JSON_VALUE(@json, '$.CustomerID'),
               @Quantity = JSON_VALUE(@json, '$.Quantity');

        SELECT @Price = Price FROM Product WHERE ProductID = @ProductID;

        INSERT INTO SalesTransaction (TransactionDate, ProductID, CustomerID,
Quantity, Total)
        VALUES (@TransactionDate, @ProductID, @CustomerID, @Quantity, @Quantity
* @Price);

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;



--Read Stored Procedure for salestransaction


CREATE PROCEDURE sp_ReadSalesTransaction
    @json NVARCHAR(MAX)
AS
BEGIN
    DECLARE @TransactionID INT;
    SELECT @TransactionID = JSON_VALUE(@json, '$.TransactionID');

    SELECT * FROM SalesTransaction WHERE TransactionID = @TransactionID;
END;



--Update Stored Procedure for salestransaction


CREATE PROCEDURE sp_UpdateSalesTransaction
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @TransactionID INT;
        DECLARE @TransactionDate DATE;
        DECLARE @ProductID INT;
        DECLARE @CustomerID INT;
        DECLARE @Quantity INT;
        DECLARE @Price DECIMAL(10,2);

        SELECT @TransactionID = JSON_VALUE(@json, '$.TransactionID'),
               @TransactionDate = JSON_VALUE(@json, '$.TransactionDate'),
               @ProductID = JSON_VALUE(@json, '$.ProductID'),
               @CustomerID = JSON_VALUE(@json, '$.CustomerID'),
               @Quantity = JSON_VALUE(@json, '$.Quantity');

        SELECT @Price = Price FROM Product WHERE ProductID = @ProductID;

        UPDATE SalesTransaction
        SET TransactionDate = @TransactionDate,
            ProductID = @ProductID,
            CustomerID = @CustomerID,
            Quantity = @Quantity,
            Total = @Quantity * @Price
        WHERE TransactionID = @TransactionID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;



--Delete Stored procedure for salestransaction


CREATE PROCEDURE sp_DeleteSalesTransaction
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @TransactionID INT;
        SELECT @TransactionID = JSON_VALUE(@json, '$.TransactionID');

        DELETE FROM SalesTransaction WHERE TransactionID = @TransactionID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH;
END;
```

CRUD Operations for INVOICE:

```
--Create Stored Procedure for Invoice


CREATE PROCEDURE sp_CreateInvoice
```

```
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @CustomerID INT;
        DECLARE @InvoiceDate DATE;

        SELECT @CustomerID = JSON_VALUE(@json, '$.CustomerID'),
               @InvoiceDate =JSON_VALUE(@json, '$.InvoiceDate')

        DECLARE @TotalBillAmount DECIMAL(10,2);
        SELECT @TotalBillAmount = SUM(Total) FROM SalesTransaction WHERE
CustomerID = @CustomerID;

        -- Apply discount
        SET @TotalBillAmount = @TotalBillAmount * CASE
                                                     WHEN @TotalBillAmount <=
1000 THEN 0.95
                                                     ELSE 0.90
                                                    END;

        INSERT INTO Invoice (InvoiceDate, TotalBillAmount)
        VALUES (@InvoiceDate, @TotalBillAmount);

        DECLARE @InvoiceID INT = SCOPE_IDENTITY();

        INSERT INTO InvoiceTransaction (InvoiceID, TransactionID)
        SELECT @InvoiceID, TransactionID
        FROM SalesTransaction
        WHERE CustomerID = @CustomerID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;


--Read Stored Procedure for Invoice


CREATE PROCEDURE sp_ReadInvoice
    @json NVARCHAR(MAX)
AS
BEGIN
    DECLARE @InvoiceID INT;
    SELECT @InvoiceID = JSON_VALUE(@json, '$.InvoiceID');

    SELECT * FROM Invoice WHERE InvoiceID = @InvoiceID;
END;



--Update Stored procedure for Invoice


CREATE PROCEDURE sp_UpdateInvoice
    @json NVARCHAR(MAX)
AS
```

```
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @InvoiceID INT;
        DECLARE @InvoiceDate DATE;

        SELECT @InvoiceID = JSON_VALUE(@json, '$.InvoiceID'),
               @InvoiceDate = JSON_VALUE(@json, '$.InvoiceDate');

        DECLARE @TotalBillAmount DECIMAL(10,2);
        SELECT @TotalBillAmount = SUM(Total)
        FROM SalesTransaction
        WHERE TransactionID IN (
            SELECT TransactionID
            FROM InvoiceTransaction
            WHERE InvoiceID = @InvoiceID
        );

        -- Apply discount

        SET @TotalBillAmount = @TotalBillAmount * CASE
                                                     WHEN @TotalBillAmount <=
1000 THEN 0.95
                                                     ELSE 0.90
                                                    END;

        UPDATE Invoice
        SET InvoiceDate = @InvoiceDate,
            TotalBillAmount = @TotalBillAmount
        WHERE InvoiceID = @InvoiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;


--Delete Stored Procedure for Invoice


CREATE PROCEDURE sp_DeleteInvoice
    @json NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @InvoiceID INT;
        SELECT @InvoiceID = JSON_VALUE(@json, '$.InvoiceID');

        DELETE FROM Invoice WHERE InvoiceID = @InvoiceID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
```

```
END
```

Q.3. Query to return following:
1. List of customers whose name starts with the letter "A" or ends with the letter "S" but
should have the letter "K".
2. Customers whose invoice is not processed yet.
3. Name of customer who has spent highest amount in a specific date range.
4. Remove the product which is not bought in the current year.
5. The product should have a remaining column which shows the remaining quantity of
the product. This should be updated on the basis of sales transactions. List out the
products whose remaining quantity is less than 2.
6. Get the product of the year (The product that was bought by maximum customers this
year.)
7. Return the list of customers who bought more than 10 products.

1).
List of customers whose name starts with the letter "A" or ends with the letter "S" but
should have the letter "K".

```
SELECT *
FROM Customer
WHERE (Name LIKE 'A%' OR Name LIKE '%S') AND Name LIKE '%K%';
```

2).
Customers whose invoice is not processed yet.

```
SELECT DISTINCT c.CustomerID, c.Name, c.Email, c.Phone
FROM Customer c
INNER JOIN SalesTransaction st ON c.CustomerID = st.CustomerID
LEFT JOIN InvoiceTransaction it ON st.TransactionID = it.TransactionID
WHERE it.TransactionID IS NULL;
```

3).
Name of customer who has spent highest amount in a specific date range.

```
DECLARE @StartDate DATE = '2021-01-02';
DECLARE @EndDate DATE = '2022-04-02';

;WITH TotalSpent AS (
    SELECT
        c.Name,
        SUM(st.Total) as TotalAmount
    FROM
        Customer c
    JOIN
        SalesTransaction st
    ON
        c.CustomerID = st.CustomerID
    WHERE
        st.TransactionDate BETWEEN @StartDate AND @EndDate
    GROUP BY
        c.Name
)
SELECT
```

```
    Name
FROM
    TotalSpent
WHERE
    TotalAmount = (SELECT MAX(TotalAmount) FROM TotalSpent);
```

4).
Remove the product which is not bought in the current year.

```
DECLARE @CurrentYear INT = YEAR(GETDATE());

DELETE FROM Product
WHERE ProductID NOT IN (
    SELECT DISTINCT ProductID
    FROM SalesTransaction
    WHERE YEAR(TransactionDate) = @CurrentYear
);
```

5).
The product should have a remaining column which shows the remaining quantity of the product. This should be updated on the basis of sales transactions. List out the
products whose remaining quantity is less than 2.

```
SELECT
    P.ProductID,
    P.ProductName,
    P.Description,
    P.Price,
    P.RemainingQuantity - ISNULL(SUM(ST.Quantity), 0) AS RemainingQuantity
FROM
    Product P
LEFT JOIN
    SalesTransaction ST ON P.ProductID = ST.ProductID
GROUP BY
    P.ProductID, P.ProductName, P.Description, P.Price, P.RemainingQuantity
HAVING
    P.RemainingQuantity - ISNULL(SUM(ST.Quantity), 0) < 2;
```

6).
Get the product of the year (The product that was bought by maximum customers this
year.)

```
SELECT TOP 1
    P.ProductID,
    P.ProductName,
    COUNT(DISTINCT ST.CustomerID) as NumberOfCustomers
FROM
    Product P
JOIN
    SalesTransaction ST ON P.ProductID = ST.ProductID
WHERE
    YEAR(ST.TransactionDate) = YEAR(GETDATE())
GROUP BY
    P.ProductID, P.ProductName
ORDER BY
    NumberOfCustomers DESC;
```

7).
Return the list of customers who bought more than 10 products.

```
SELECT
    C.CustomerID,
    C.Name,
    COUNT(DISTINCT ST.ProductID) as NumberOfProducts
FROM
    Customer C
JOIN
    SalesTransaction ST ON C.CustomerID = ST.CustomerID
GROUP BY
    C.CustomerID, C.Name
HAVING
    COUNT(DISTINCT ST.ProductID) > 10;
```

Q.4
Create a function with following:
• Input Parameters:
i)Customer (can pass multiple ids e.g.: 103,904)
ii)Start Date and End Date
• Requirement:
i)Should return total bill amount of the customer in the given date range.

At first, Creating a type to hold table of customerIDs:

```
CREATE TYPE CustomerIDTableType AS TABLE
(
    CustomerID INT PRIMARY KEY
);
```

And, Creating the function:

```
CREATE FUNCTION GetTotalBillAmount
(
    @CustomerIDs CustomerIDTableType READONLY,
    @StartDate DATE,
    @EndDate DATE
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        C.CustomerID,
        SUM(ST.Total) AS TotalBillAmount
    FROM
        Customer C
    INNER JOIN
        SalesTransaction ST ON C.CustomerID = ST.CustomerID
    INNER JOIN
        @CustomerIDs T ON C.CustomerID = T.CustomerID
    WHERE
        ST.TransactionDate BETWEEN @StartDate AND @EndDate
    GROUP BY
        C.CustomerID
);
```

And, Finally calling this function by passing a table of customer IDs and a date range:

```
DECLARE @CustomerIDs CustomerIDTableType;

INSERT INTO @CustomerIDs (CustomerID)
VALUES (103), (904);

SELECT * FROM GetTotalBillAmount(@CustomerIDs, '2022-01-01', '2022-12-31');
```

Q.5
Create a Store Procedure:
• Input Parameters:
• Start Date and End date
• Customer Id (if no customer Id is passed then return all data else the given customer’s
data only)
• Requirement:
• Should return all the customers’ information entered in the database within the date
range
• Should return the total invoice (bill) amount of the customer

Use of Json is mandatory for input parameter and to return values.

```
CREATE PROCEDURE sp_GetCustomerInfo
    @json NVARCHAR(MAX)
AS
BEGIN

    DECLARE @StartDate DATE, @EndDate DATE, @CustomerId INT;


    SELECT
        @StartDate = JSON_VALUE(@json, '$.StartDate'),
        @EndDate = JSON_VALUE(@json, '$.EndDate'),
        @CustomerId = JSON_VALUE(@json, '$.CustomerId');

    -- Query to fetch customer information and total invoice amount
    SELECT
        C.CustomerID,
        C.Name,
        C.Email,
        C.Phone,
        SUM(I.TotalBillAmount) AS TotalInvoiceAmount
    FROM
        Customer C
        INNER JOIN SalesTransaction ST ON C.CustomerID = ST.CustomerID
        INNER JOIN InvoiceTransaction IT ON ST.TransactionID = IT.TransactionID
        INNER JOIN Invoice I ON IT.InvoiceID = I.InvoiceID
    WHERE
        (C.CustomerID = @CustomerId OR @CustomerId IS NULL) AND
        (I.InvoiceDate BETWEEN @StartDate AND @EndDate)
    GROUP BY
        C.CustomerID,
        C.Name,
        C.Email,
        C.Phone
    FOR JSON PATH; --allow to format result of queries as JSON
END;
```