Case Study – Data Engineering Role

1. Data Transformation:

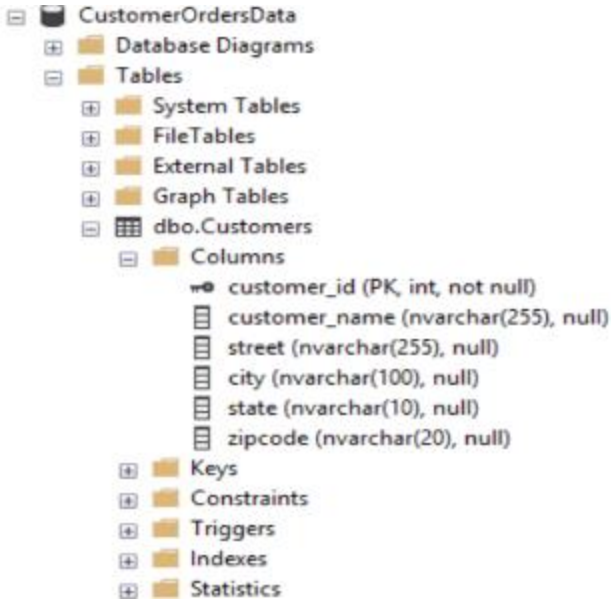
Json file structure

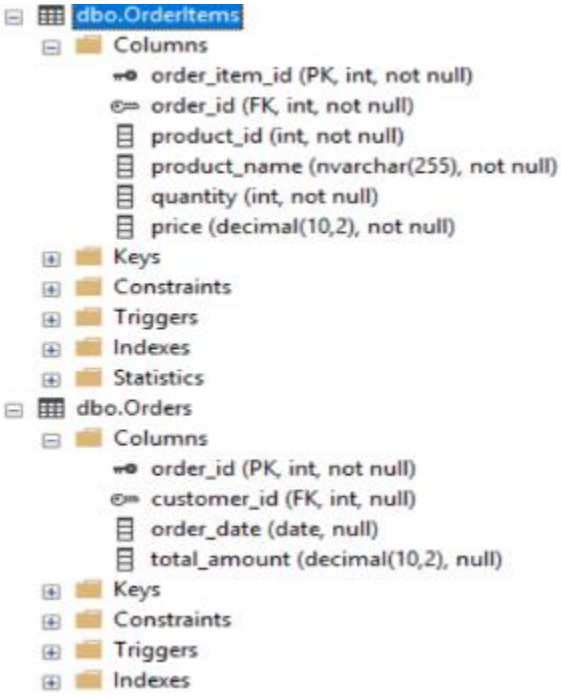

```
[
  {
    "customer_id": 12345,
    "customer_name": "John Doe",
    "address": {
      "street": "123 Elm St",
      "city": "Springfield",
      "state": "IL",
      "zipcode": "62701"
    },
    "orders": [
      {
        "order_id": 1,
        "order_date": "2025-02-01",
        "total_amount": 150.00,
        "items": [
          {
            "product_id": 101,
            "product_name": "Laptop",
            "quantity": 1,
            "price": 1200.00
          },
          {
            "product_id": 102,
            "product_name": "Mouse",
            "quantity": 2,
            "price": 25.00
          }
        ]
      },
      {
        "order_id": 2,
        "order_date": "2025-02-05",
        "total_amount": 200.00,
        "items": [
          {
            "product_id": 103,
```

```
                "product_name": "Keyboard",
                "quantity": 1,
                "price": 50.00
            }
        ]
    }
  ]
 }
]
```

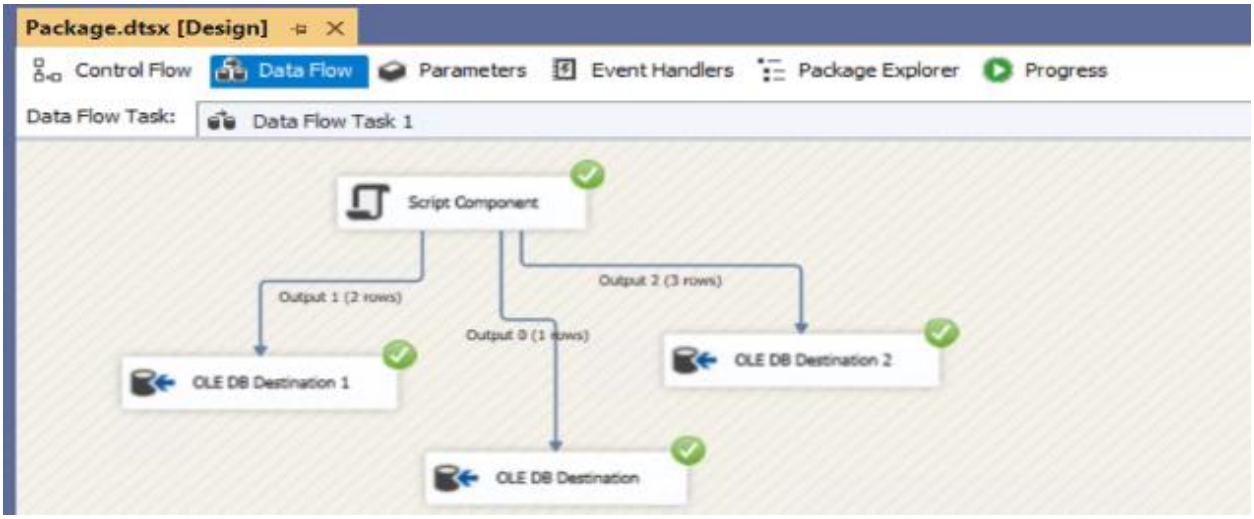
Create the Database structure

```
CREATE DATABASE CustomerOrdersData;

USE [CustomerOrdersData];

CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    customer_name NVARCHAR(255),
    street NVARCHAR(255),
    city NVARCHAR(100),
    state NVARCHAR(10),
    zipcode NVARCHAR(20)
);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
CREATE TABLE OrderItems (
    order_item_id INT IDENTITY(1,1) PRIMARY KEY,  -- Unique ID for each order item
    order_id INT,                                  -- Foreign key referencing the Orders
table
    product_id INT,                                -- Product ID for the item
    product_name NVARCHAR(255),                    -- Product name
    quantity INT,                                  -- Quantity of the product ordered
    price DECIMAL(10, 2),                          -- Price of the product
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)  -- Foreign key constraint
);
```





My ETL Design



Flattening and Transformation

Objective of Flattening:

- Transform complex nested objects and arrays into simpler, single-level records for each entity (e.g., flattening nested orders, items, or address data).
- This step is essential because relational databases typically do not handle nested data well, so the nested elements need to be extracted and treated as separate tables or flattened fields.

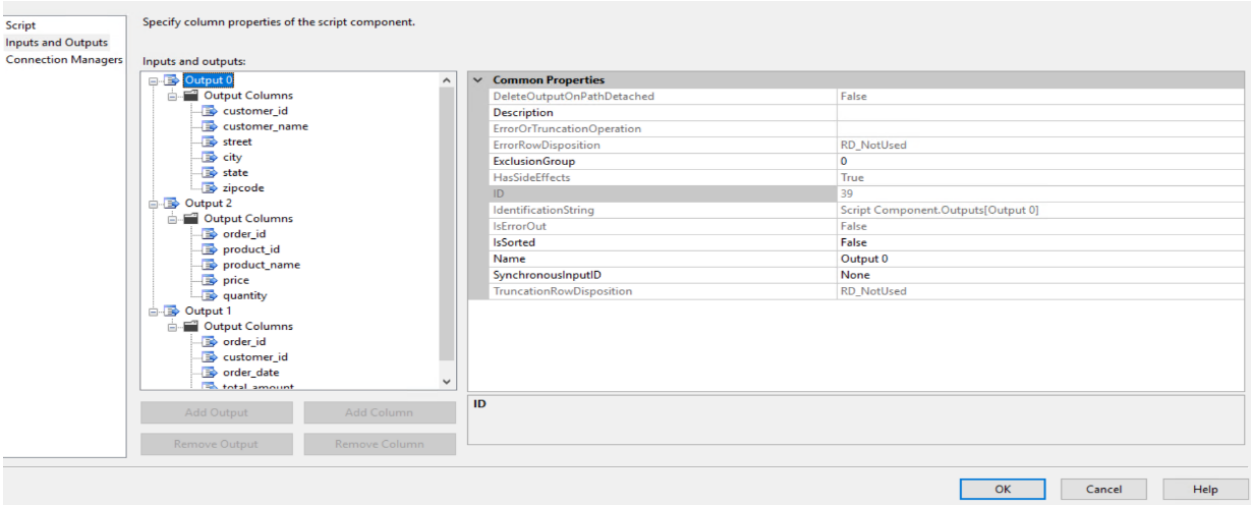
we need to loop through each object, extract data from the nested structure, and populate it in a way that aligns with the relational schema.

C# Code:

1. **Read the JSON File**: Load the JSON content from the file and deserialize it into a C# object.
2. **Flattening**: Flatten the nested fields (such as address, orders, and items), and store each of these pieces of data in their respective tables.

Script component

Inputs and Outputs



This part of my script

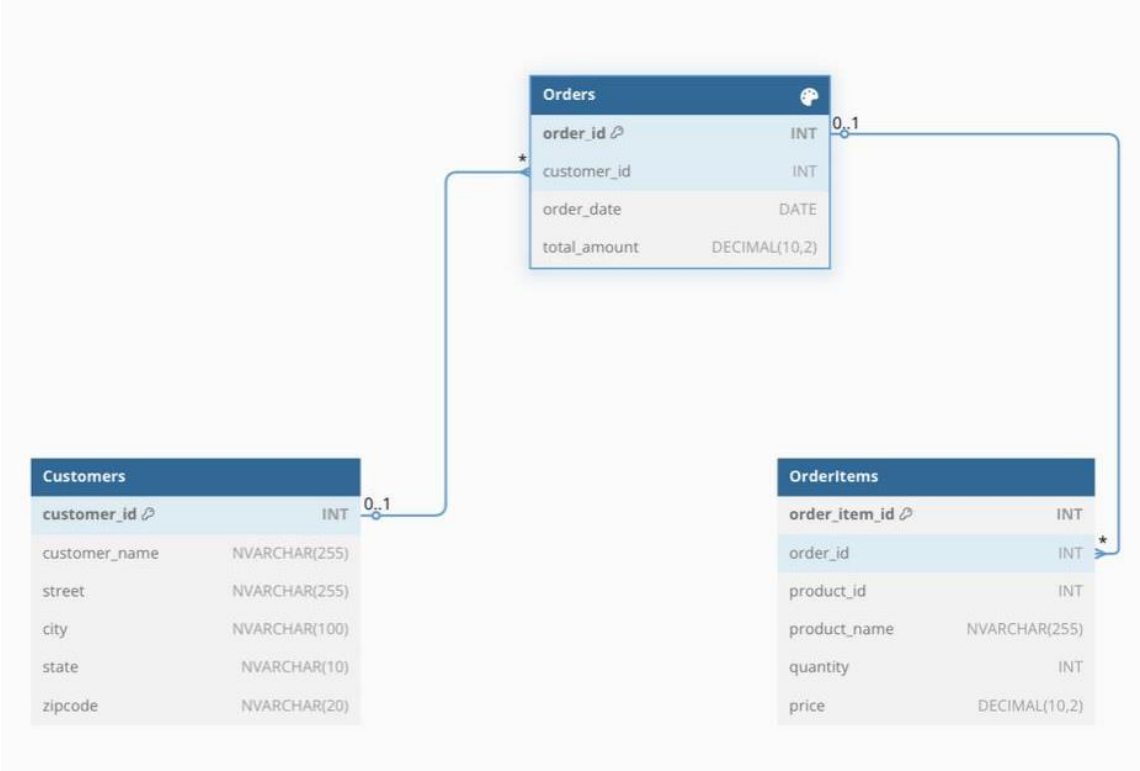
```
public class Customer
 {
     public int customer_id { get; set; }
     public string customer_name { get; set; }
     public Address address { get; set; }
     public List<Order> orders { get; set; }
 }

 public class Address
 {
     public string street { get; set; }
     public string city { get; set; }
     public string state { get; set; }
     public string zipcode { get; set; }
 }

 public class Order
 {
     public int order_id { get; set; }
     public string order_date { get; set; }  // Keeping it as string for parsing
     public decimal total_amount { get; set; }
     public List<OrderItem> items { get; set; }
 }

 public class OrderItem
 {
     public int product_id { get; set; }
     public string product_name { get; set; }
     public int quantity { get; set; }
     public decimal price { get; set; }
 }
```

CreateNewOutputRows() method to flatten and normalize the data

```
public override void CreateNewOutputRows()
{
    String jsonFileContent = File.ReadAllText(@"C:\Users\adm-
mohamsha02\Documents\CustomerOrders.json");
    JavaScriptSerializer js = new JavaScriptSerializer();
    List<Customer> customers = js.Deserialize<List<Customer>>(jsonFileContent);

    // Loop through customers and add data to buffers
    foreach (Customer customer in customers)
    {
        //  ◇  Customers Output (Configured Data Types)
        Output0Buffer.AddRow();
        Output0Buffer.customerid = customer.customer_id;  // INT
        Output0Buffer.customername = customer.customer_name ?? ""; ;  //
NVARCHAR(255)
        Output0Buffer.street = customer.address?.street ?? "";  // NVARCHAR(255)
        Output0Buffer.city = customer.address?.city ?? "";  // NVARCHAR(100)
        Output0Buffer.state = customer.address?.state ?? "";  // NVARCHAR(10)
        Output0Buffer.zipcode = customer.address?.zipcode ?? "";  // NVARCHAR(20)

        // Loop through Orders
        foreach (Order order in customer.orders)
        {
            //  ◇  Orders Output (Configured Data Types)
            Output1Buffer.AddRow();
            Output1Buffer.orderid = order.order_id;  // INT
            Output1Buffer.customerid = customer.customer_id;  // INT
            Output1Buffer.orderdate = DateTime.TryParse(order.order_date, out
DateTime parsedDate) ? parsedDate : DateTime.MinValue;  // DATE
            Output1Buffer.totalamount = order.total_amount;  // DECIMAL(10,2)

            // Loop through Order Items
            foreach (OrderItem item in order.items)
            {
                //  ◇  OrderItems Output (Configured Data Types)
                Output2Buffer.AddRow();
                Output2Buffer.orderid = order.order_id;  // INT (Foreign Key)
                Output2Buffer.productid = item.product_id;  // INT
                Output2Buffer.productname = item.product_name ?? "";  //
NVARCHAR(255)
                Output2Buffer.quantity = item.quantity;  // INT
                Output2Buffer.price = item.price;  // DECIMAL(10,2)
            }



        }
    }
}
```

- After flattening and normalizing, use the **DB Destination Component** (e.g., OLE DB Destination in SSIS or Database Output in Talend) to insert the data into the relational database.

2. Explanation of the Database Structure



1. Normalization in relational databases involves designing a schema where the data is stored in separate tables to avoid redundancy and ensure consistency.

In this case:

- **Customer table**: Contains only customer-related information (e.g., `customer_id`, `customer_name`, and address).
- **Orders table**: Contains details of each order, linked by `customer_id` to the customer who placed it.
- **OrderItems table**: Stores individual items within each order, linked by `order_id`.

By doing this we achieve:

- Each piece of data is stored only once, preventing redundancy.
- Ensure that updates, deletions, and insertions happen without anomalies.
- Relationships between customers, orders, and items are clearly defined using foreign keys (e.g., `customer_id` in the `Orders` table, `order_id` in the `OrderItems` table).

2. **Relationships**:
    - **One-to-Many**: One customer can have multiple orders, so the `Orders` table has a foreign key referencing `Customers`.
    - **One-to-Many**: One order can have multiple items, so the `OrderItems` table has a foreign key referencing `Orders`.

3. **Data Integrity**:
    a. **Foreign Key Constraints** ensure that each order is linked to an existing customer and each order item is linked to an existing order.

3. Database Optimization

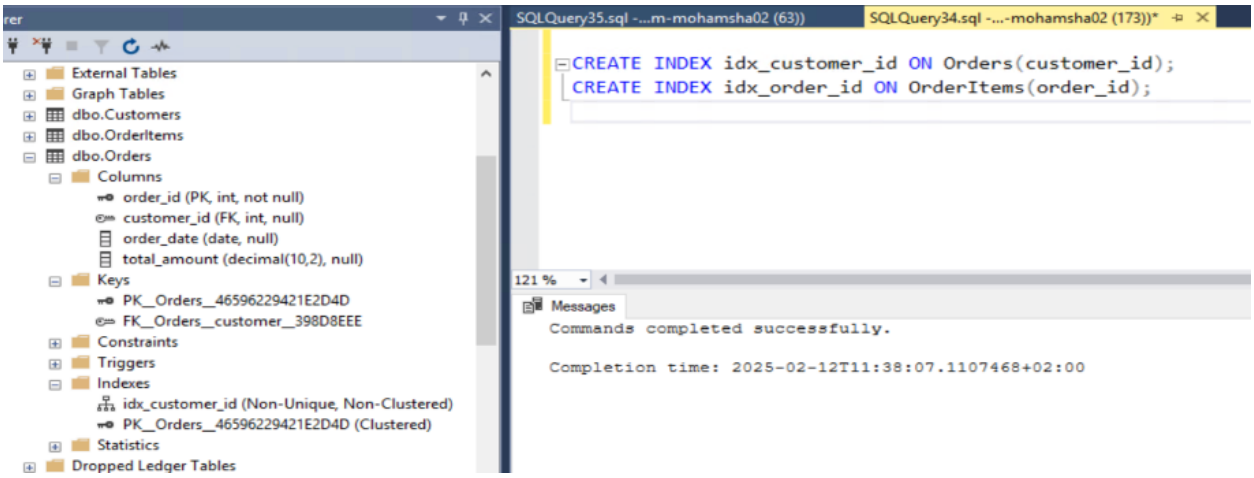
To enhance query performance, you can add indexes on foreign key columns and other frequently queried columns. For example:

-- Add an index on customer_id in the Orders table

CREATE INDEX idx_customer_id ON Orders(customer_id);

-- Add an index on order_id in the OrderItems table

CREATE INDEX idx_order_id ON OrderItems(order_id);

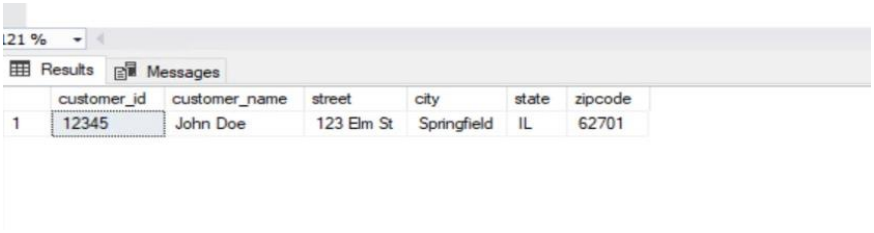


Also, we can use partitioning If `Orders` table grows, partition by `order_date` for better query speed.

```
CREATE PARTITION FUNCTION pfOrders (DATE)
AS RANGE RIGHT FOR VALUES ('2025-01-01', '2025-07-01');

CREATE PARTITION SCHEME psOrders
AS PARTITION pfOrders ALL TO ([PRIMARY]);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2),
    INDEX idx_order_date(order_date)
) ON psOrders(order_date);
```

4. Output Results

1. Customer Table



2. Orders Table

SQLQuery34.sql -...-mohamsha02 (173)) | SQLQuery33.sql -...-mohamsha02 (175))*

```
SELECT TOP (1000) [order_id]
      ,[customer_id]
      ,[order_date]
      ,[total_amount]
  FROM [CustomerOrdersData].[dbo].[Orders]
```

121 %

Results | Messages

	order_id	customer_id	order_date	total_amount
1	1	12345	2025-02-01	150.00
2	2	12345	2025-02-05	200.00

3. Order Items

SQLQuery35.sql -...m-mohamsha02 (63)) | SQLQuery34.sql -...-mohamsha02 (173)) | SQLQu

```
SELECT TOP (1000) [order_item_id]
      ,[order_id]
      ,[product_id]
      ,[product_name]
      ,[quantity]
      ,[price]
  FROM [CustomerOrdersData].[dbo].[OrderItems]
```

121 %

Results | Messages

	order_item_id	order_id	product_id	product_name	quantity	price
1	1	1	101	Laptop	1	1200.00
2	2	1	102	Mouse	2	25.00
3	3	2	103	Keyboard	1	50.00

Another approach could be done using the Json extraction like this, but it is not flexible and limited to small data

```
-- Load JSON data from file
SELECT @JSON = BulkColumn
FROM OPENROWSET (BULK 'C:\Users\adm-mohamsha02\Documents\Json_test.js', SINGLE_CLOB) AS import;
-- Create table for customer information
CREATE TABLE Customer (
  CustomerID INT,
  CustomerName NVARCHAR(100),
  Street NVARCHAR(100),
  City NVARCHAR(50),
  State NVARCHAR(50),
  Zipcode NVARCHAR(10)
);

-- Create table for orders
CREATE TABLE Orders (
  OrderID INT,
  CustomerID INT,
  OrderDate DATE,
  TotalAmount DECIMAL(10, 2)
);
```

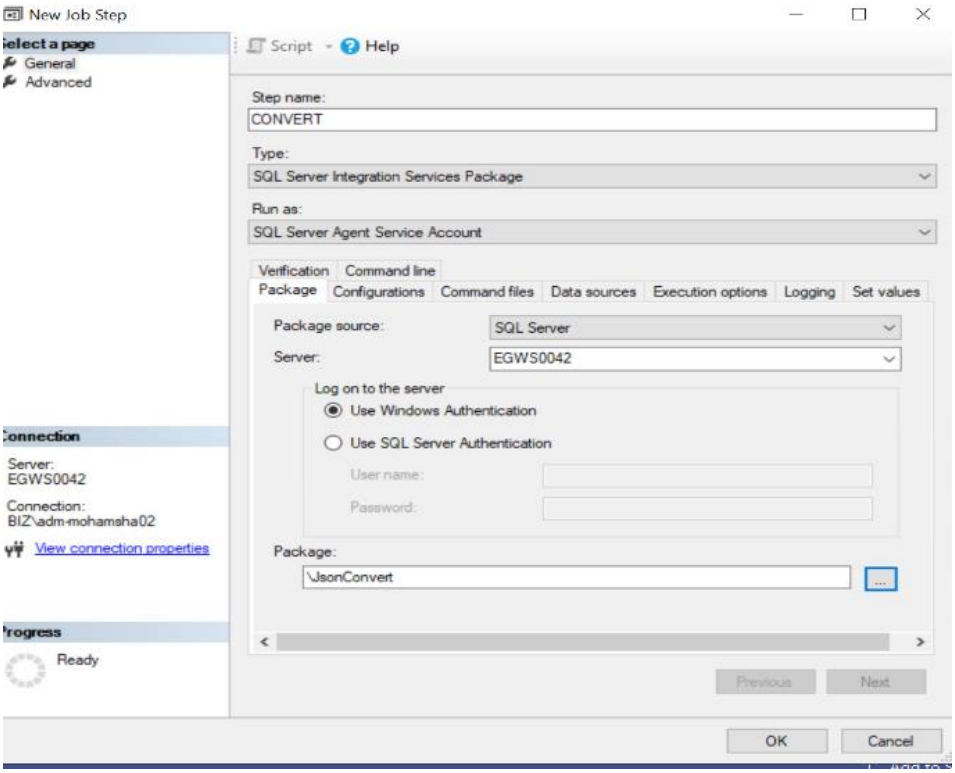
Limitations

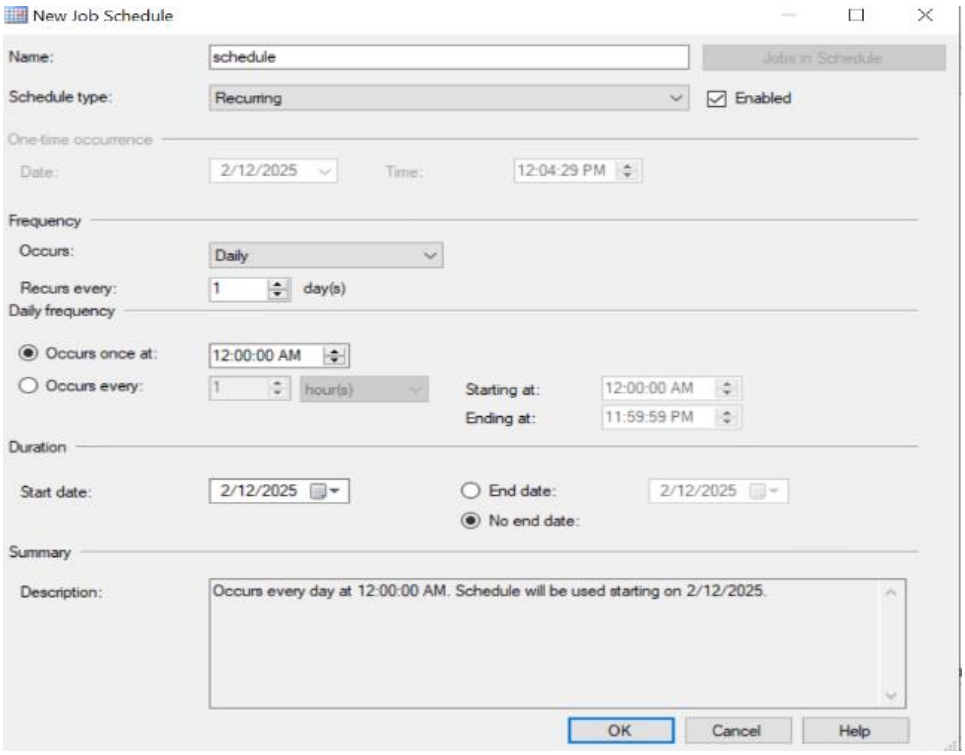
Bulk Insert Performance Issues

- OPENROWSET loads the entire JSON into memory, which **does not scale well for large files**.
- You must manually parse and insert nested data, which is **inefficient for large datasets**.
- **For very large JSON files, this approach may lead to memory exhaustion.**

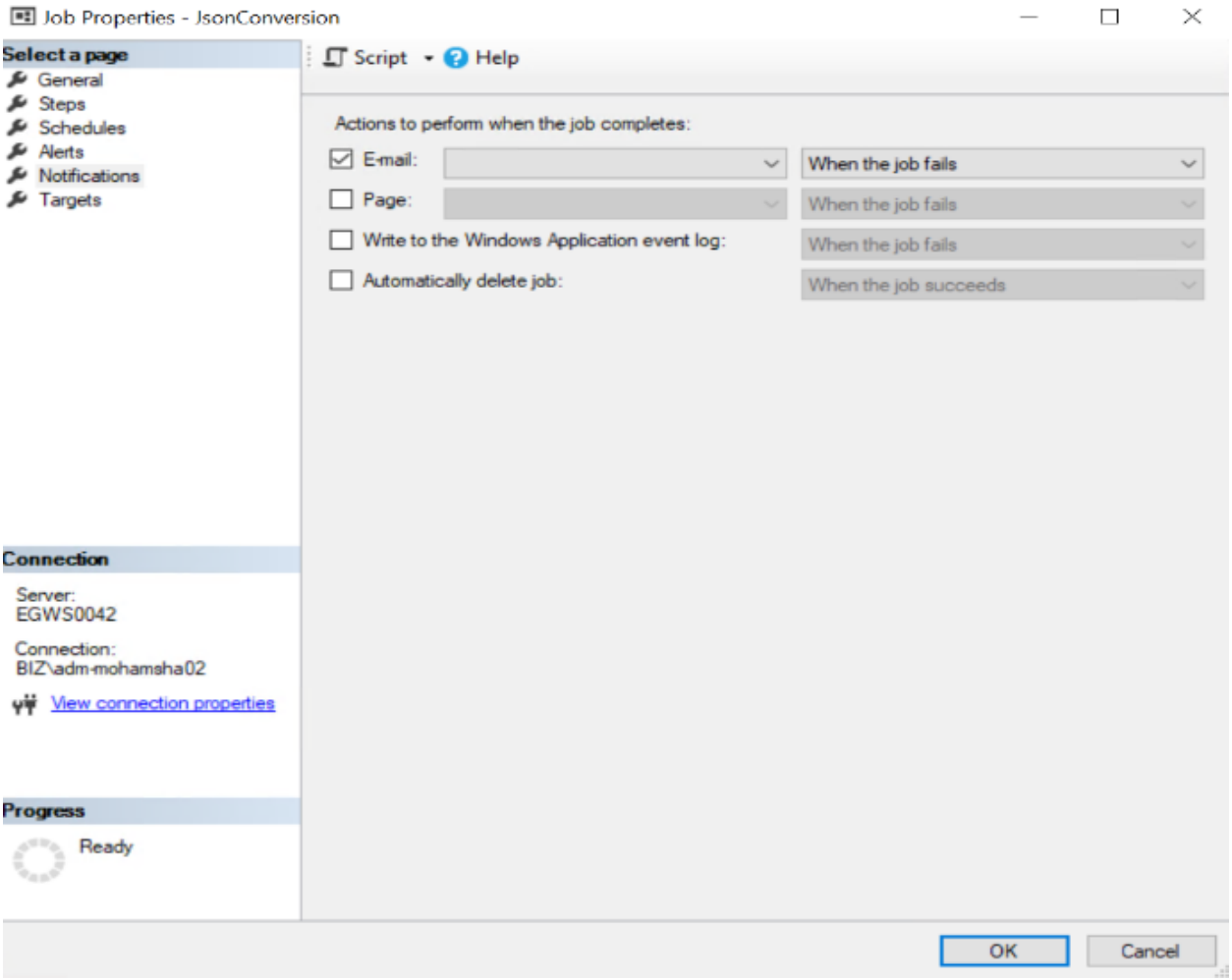
5 . Job Notification

Creating Job and schedule the time of running





Send notification when job fails to my email



GitHub Link

https://github.com/Shaimaa-moh/JsonToTables/tree/main/JsonTaskTrial2%20-%20Copy%20(2)