FULL Outer Join

-- Create Employees Table

CREATETABLEEmployees (

EmployeeIDINTPRIMARYKEY,

EmployeeNameVARCHAR(100),

DepartmentIDINT

);

-- Insert Data into Employees Table

INSERTINTOEmployees (EmployeeID,EmployeeName,DepartmentID)VALUES (1,'John Doe',1);

INSERTINTOEmployees (EmployeeID,EmployeeName,DepartmentID)VALUES (2,'Jane Smith',2);

INSERTINTOEmployees (EmployeeID,EmployeeName,DepartmentID)VALUES (3,'Mark Brown',3);

INSERTINTOEmployees (EmployeeID,EmployeeName,DepartmentID)VALUES (4,'Lucy Green',NULL);

-- Create Departments Table

CREATETABLEDepartments (

DepartmentIDINTPRIMARYKEY,

DepartmentNameVARCHAR(100)

);

-- Insert Data into Departments Table

INSERTINTODepartments (DepartmentID,DepartmentName)VALUES (1,'HR');

INSERTINTODepartments (DepartmentID,DepartmentName)VALUES (2,'Sales');

INSERTINTODepartments (DepartmentID,DepartmentName)VALUES (4,'IT');

SELECT\*FROMEmployees

SELECT\*FROMDepartments

SELECT

Employees.EmployeeID,

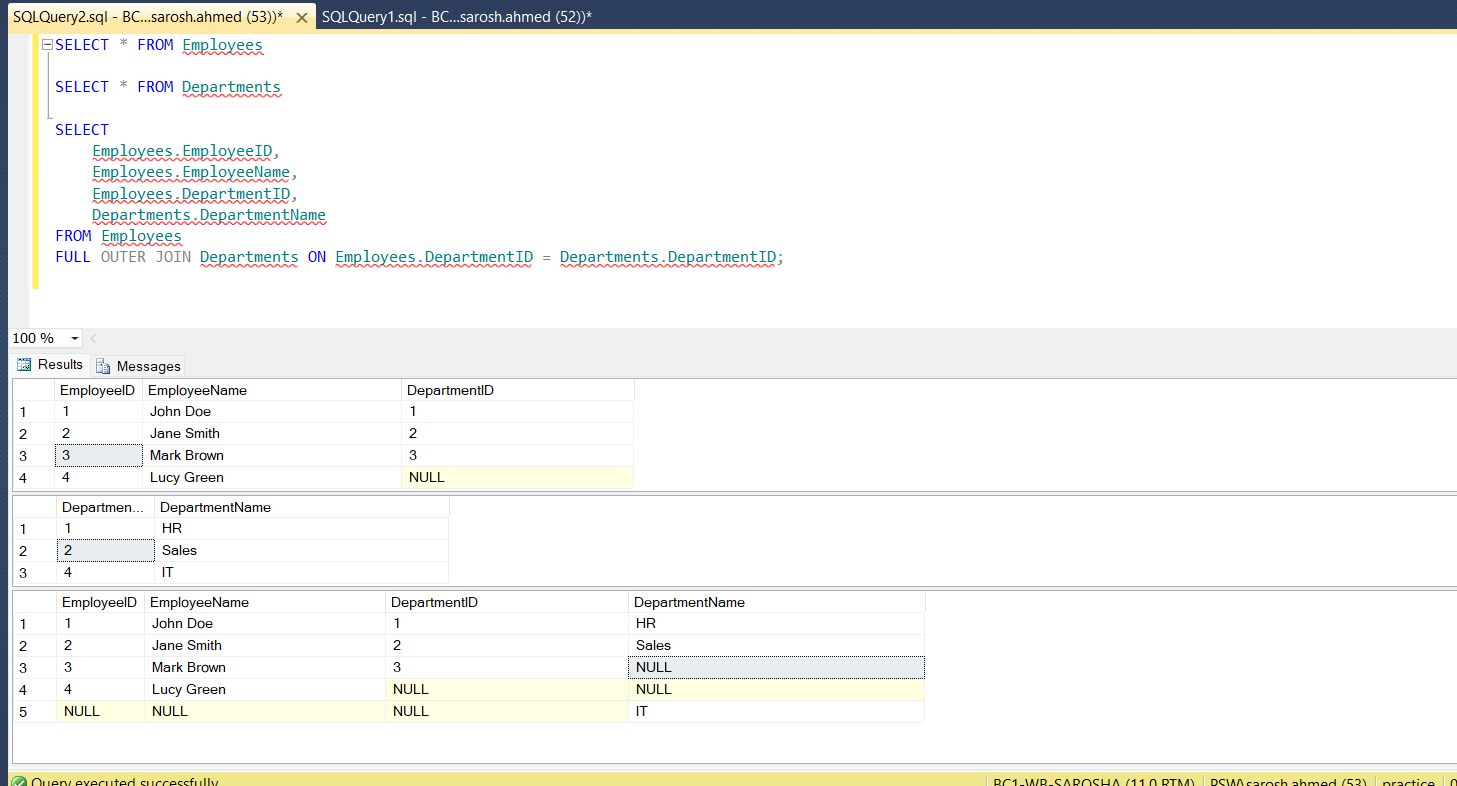
Employees.EmployeeName,

Employees.DepartmentID,

Departments.DepartmentName

FROMEmployees

FULLOUTERJOINDepartmentsONEmployees.DepartmentID=Departments.DepartmentID;



-- Create Customers Table

CREATETABLECustomers (

CustomerIDINTPRIMARYKEY,

CustomerNameVARCHAR(100)

);

-- Insert Data into Customers Table

INSERTINTOCustomers (CustomerID,CustomerName)VALUES (1,'Alice');

INSERTINTOCustomers (CustomerID,CustomerName)VALUES (2,'Bob');

INSERTINTOCustomers (CustomerID,CustomerName)VALUES (3,'Charlie');

INSERTINTOCustomers (CustomerID,CustomerName)VALUES (4,'David');

-- Create Orders Table

CREATETABLEOrders (

OrderIDINTPRIMARYKEY,

CustomerIDINT,

OrderAmountDECIMAL(10,2),

FOREIGNKEY (CustomerID)REFERENCESCustomers(CustomerID)

);

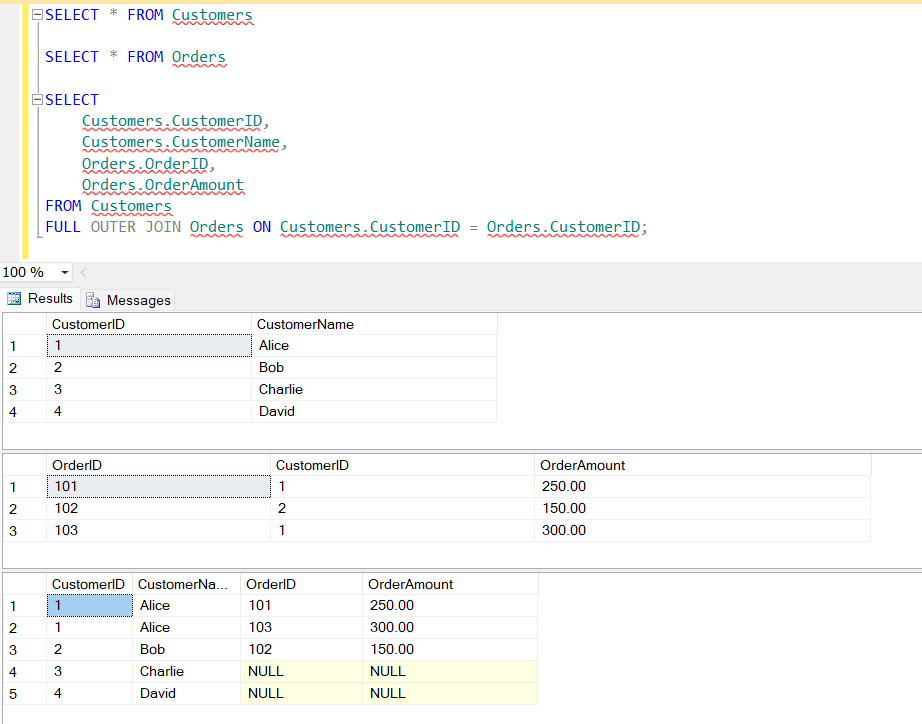
-- Insert Data into Orders Table

INSERTINTOOrders (OrderID,CustomerID,OrderAmount)VALUES (101,1,250.00);

INSERTINTOOrders (OrderID,CustomerID,OrderAmount)VALUES (102,2,150.00);

INSERTINTOOrders (OrderID,CustomerID,OrderAmount)VALUES (103,1,300.00);

-- Removed the problematic insert statement with CustomerID 5



SELF JOIN

-- Create Employees Table

CREATETABLEEmployees (

EmployeeIDINTPRIMARYKEY,

EmployeeNameVARCHAR(100),

ManagerIDINT

);

-- Insert Data into Employees Table

INSERTINTOEmployees (EmployeeID,EmployeeName,ManagerID)VALUES (1,'John Doe',NULL);

INSERTINTOEmployees (EmployeeID,EmployeeName,ManagerID)VALUES (2,'Jane Smith',1);

INSERTINTOEmployees (EmployeeID,EmployeeName,ManagerID)VALUES (3,'Mark Brown',1);

INSERTINTOEmployees (EmployeeID,EmployeeName,ManagerID)VALUES (4,'Lucy Green',2);

INSERTINTOEmployees (EmployeeID,EmployeeName,ManagerID)VALUES (5,'David White',3);

SELECT\*FROMEmployees

SELECT

E1.EmployeeIDASEmployeeID,

E1.EmployeeNameASEmployeeName,

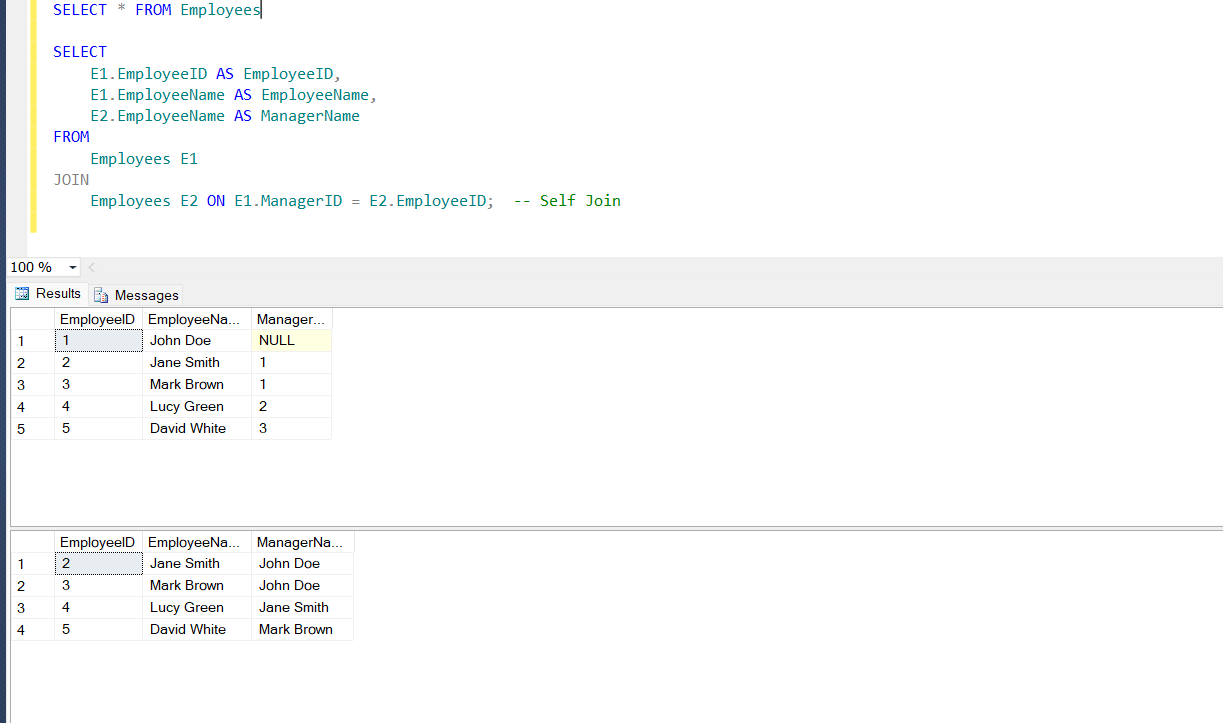
E2.EmployeeNameASManagerName

FROM

EmployeesE1

JOIN

EmployeesE2ONE1.ManagerID=E2.EmployeeID;-- Self Join



**Self joins** can be applied in a variety of real-world scenarios where you need to compare rows within the same table or establish relationships within the same dataset. Here are some practical examples:

**1. Employee Hierarchy (Manager-Employee Relationships)**

This is a common use case where you need to display hierarchical relationships between employees and their managers.

**Scenario:**

You have an Employees table where each employee has a ManagerID that references another employee's EmployeeID. A self join can help list all employees along with their manager's name.

**Example Query:**

sql

Copy code

SELECT

E1.EmployeeID AS EmployeeID,

E1.EmployeeName AS EmployeeName,

E2.EmployeeName AS ManagerName

FROM

Employees E1

JOIN

Employees E2 ON E1.ManagerID = E2.EmployeeID;

**2. Finding Duplicate Records**

Self joins can be used to find duplicate records in a table based on one or more column values.

**Scenario:**

You have a Customers table, and you want to identify all customers who have the same Email or PhoneNumber.

**Example Table: Customers**

| **CustomerID** | **CustomerName** | **Email** | **PhoneNumber** |
| --- | --- | --- | --- |
| 1 | Alice | alice@example.com | 1234567890 |
| 2 | Bob | bob@example.com | 1234567890 |
| 3 | Charlie | alice@example.com | 9876543210 |
| 4 | David | david@example.com | 1111111111 |

**Example Query:**

sql

Copy code

SELECT

C1.CustomerID,

C1.CustomerName,

C1.Email,

C1.PhoneNumber

FROM

Customers C1

JOIN

Customers C2 ON C1.Email = C2.Email OR C1.PhoneNumber = C2.PhoneNumber

WHERE

C1.CustomerID < C2.CustomerID; -- Prevent duplicate pairs in the result

**3. Matching Products Based on Attributes**

You can use a self join to match products based on similar attributes, like matching products with the same category or price range.

**Scenario:**

You have a Products table where each product belongs to a category. You want to find pairs of products within the same category but with different prices.

**Example Table: Products**

| **ProductID** | **ProductName** | **Category** | **Price** |
| --- | --- | --- | --- |
| 1 | Product A | Electronics | 100 |
| 2 | Product B | Electronics | 150 |
| 3 | Product C | Furniture | 200 |
| 4 | Product D | Electronics | 200 |

**Example Query:**

sql

Copy code

SELECT

P1.ProductName AS Product1,

P2.ProductName AS Product2,

P1.Category,

P1.Price AS Price1,

P2.Price AS Price2

FROM

Products P1

JOIN

Products P2 ON P1.Category = P2.Category AND P1.ProductID < P2.ProductID; -- Prevent duplicate pairs

**4. Parent-Child Relationships in Organizational Charts**

When you need to display parent-child relationships, such as organizational charts, you can use a self join to retrieve managers and their direct reports.

**Scenario:**

You have an Organization table where each entry has a ParentDepartmentID that references another department's DepartmentID.

**Example Table: Organization**

| **DepartmentID** | **DepartmentName** | **ParentDepartmentID** |
| --- | --- | --- |
| 1 | HR | NULL |
| 2 | Recruitment | 1 |
| 3 | Training | 1 |
| 4 | IT | NULL |
| 5 | Support | 4 |

**Example Query:**

sql

Copy code

SELECT

O1.DepartmentName AS Department,

O2.DepartmentName AS ParentDepartment

FROM

Organization O1

LEFT JOIN

Organization O2 ON O1.ParentDepartmentID = O2.DepartmentID;

**5. Finding Pairs of Students for a Project**

In education or team-building contexts, you might want to pair students who have not previously been paired for a project.

**Scenario:**

You have a Students table, and you want to find all possible unique pairs of students.

**Example Table: Students**

| **StudentID** | **StudentName** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |
| 4 | David |

**Example Query:**

sql

Copy code

SELECT

S1.StudentName AS Student1,

S2.StudentName AS Student2

FROM

Students S1

JOIN

Students S2 ON S1.StudentID < S2.StudentID; -- Avoid self-pairing and duplicate pairs

**6. Finding Products with Similar Pricing**

You might want to find products in the same or similar price range, e.g., within $10 of each other.

**Example Query:**

sql

Copy code

SELECT

P1.ProductName AS Product1,

P2.ProductName AS Product2,

P1.Price,

P2.Price

FROM

Products P1

JOIN

Products P2 ON ABS(P1.Price - P2.Price) <= 10 AND P1.ProductID < P2.ProductID; -- Price range condition

**Summary**

These real-world scenarios demonstrate how **self joins** can be used to find relationships, compare records, detect duplicates, and identify matches within the same dataset. The versatility of self joins makes them a powerful tool in SQL queries, especially in cases of hierarchical or relational data.

If you have any more questions or need additional examples, feel free to ask!

4o

**You said:**

give table of code of all six examples and print table also

**ChatGPT said:**

ChatGPT

Below is the SQL code for creating and inserting data into tables for all six examples. After creating and populating each table, we also include queries to display the tables and verify the data.

**1. Employee Hierarchy (Manager-Employee Relationships)**

**Table Creation and Data Insertion:**

sql

Copy code

-- Drop the Employees table if it already exists

DROP TABLE IF EXISTS Employees;

-- Create Employees Table

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(100),

ManagerID INT

);

-- Insert Data into Employees Table

INSERT INTO Employees (EmployeeID, EmployeeName, ManagerID) VALUES (1, 'John Doe', NULL);

INSERT INTO Employees (EmployeeID, EmployeeName, ManagerID) VALUES (2, 'Jane Smith', 1);

INSERT INTO Employees (EmployeeID, EmployeeName, ManagerID) VALUES (3, 'Mark Brown', 1);

INSERT INTO Employees (EmployeeID, EmployeeName, ManagerID) VALUES (4, 'Lucy Green', 2);

INSERT INTO Employees (EmployeeID, EmployeeName, ManagerID) VALUES (5, 'David White', 3);

-- Display the Employees Table

SELECT \* FROM Employees;

**2. Finding Duplicate Records**

**Table Creation and Data Insertion:**

sql

Copy code

-- Drop the Customers table if it already exists

DROP TABLE IF EXISTS Customers;

-- Create Customers Table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

Email VARCHAR(100),

PhoneNumber VARCHAR(15)

);

-- Insert Data into Customers Table

INSERT INTO Customers (CustomerID, CustomerName, Email, PhoneNumber) VALUES (1, 'Alice', 'alice@example.com', '1234567890');

INSERT INTO Customers (CustomerID, CustomerName, Email, PhoneNumber) VALUES (2, 'Bob', 'bob@example.com', '1234567890');

INSERT INTO Customers (CustomerID, CustomerName, Email, PhoneNumber) VALUES (3, 'Charlie', 'alice@example.com', '9876543210');

INSERT INTO Customers (CustomerID, CustomerName, Email, PhoneNumber) VALUES (4, 'David', 'david@example.com', '1111111111');

-- Display the Customers Table

SELECT \* FROM Customers;

**3. Matching Products Based on Attributes**

**Table Creation and Data Insertion:**

sql

Copy code

-- Drop the Products table if it already exists

DROP TABLE IF EXISTS Products;

-- Create Products Table

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2)

);

-- Insert Data into Products Table

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES (1, 'Product A', 'Electronics', 100);

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES (2, 'Product B', 'Electronics', 150);

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES (3, 'Product C', 'Furniture', 200);

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES (4, 'Product D', 'Electronics', 200);

-- Display the Products Table

SELECT \* FROM Products;

**4. Parent-Child Relationships in Organizational Charts**

**Table Creation and Data Insertion:**

sql

Copy code

-- Drop the Organization table if it already exists

DROP TABLE IF EXISTS Organization;

-- Create Organization Table

CREATE TABLE Organization (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100),

ParentDepartmentID INT

);

-- Insert Data into Organization Table

INSERT INTO Organization (DepartmentID, DepartmentName, ParentDepartmentID) VALUES (1, 'HR', NULL);

INSERT INTO Organization (DepartmentID, DepartmentName, ParentDepartmentID) VALUES (2, 'Recruitment', 1);

INSERT INTO Organization (DepartmentID, DepartmentName, ParentDepartmentID) VALUES (3, 'Training', 1);

INSERT INTO Organization (DepartmentID, DepartmentName, ParentDepartmentID) VALUES (4, 'IT', NULL);

INSERT INTO Organization (DepartmentID, DepartmentName, ParentDepartmentID) VALUES (5, 'Support', 4);

-- Display the Organization Table

SELECT \* FROM Organization;

**5. Finding Pairs of Students for a Project**

**Table Creation and Data Insertion:**

sql

Copy code

-- Drop the Students table if it already exists

DROP TABLE IF EXISTS Students;

-- Create Students Table

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

StudentName VARCHAR(100)

);

-- Insert Data into Students Table

INSERT INTO Students (StudentID, StudentName) VALUES (1, 'Alice');

INSERT INTO Students (StudentID, StudentName) VALUES (2, 'Bob');

INSERT INTO Students (StudentID, StudentName) VALUES (3, 'Charlie');

INSERT INTO Students (StudentID, StudentName) VALUES (4, 'David');

-- Display the Students Table

SELECT \* FROM Students;

**6. Finding Products with Similar Pricing**

The Products table from example 3 can be reused here since it matches the requirements for this example. There is no need to create a new table.

sql

Copy code

-- Reuse Products Table from Example 3

-- Display the Products Table again

SELECT \* FROM Products;

**Summary**

The SQL code provided creates the tables and populates them with sample data for all six examples of self join scenarios. You can run each block of code in your SQL Server environment to set up the data and practice the self join queries. If you need further examples or explanations, let me know!