MARMARA UNIVERSITY

FACULTY OF ENGINEERING

CSE3055

PROJECT STEP 3

Database Systems

01.01.2024

	Dept	Student Id	Name Surname
1	CSE	150120022	Tolga Fehmioğlu
2	CSE	150121538	Muhammed Enes Gökdeniz
3	CSE	150121002	Enes Torluoğlu
4	CSE	150121032	Mehmet Toprak Balıkcı

LAW FIRM DATABASE
Project Step #3

Project Description:

The database project revolves around managing a law firm's operations, encompassing various aspects such as employee management, client relationships, case and trial handling and payments. It manages the organization and tracking of cases, associated clients, payments, and employee details within the firm.

Scope:

Included:

- Employee Management
  - Storing details of employees (lawyers, managers, representatives), their contact information, associations, and information about their bank accounts.
- Client Management
  - Storing client information (person or company), their contact details, and associations with cases.
- Case Management
  - Tracking case details, associated clients, relevancy periods, and lawyers handling the cases.
- Payments
  - Managing payment details, dates, transactions, and the status of debts.
- Trials
  - Organizing trial details, associations with cases, and representatives involved.

Excluded:

- Detailed Financial Account
  - While payment details and debts are tracked, complex financial calculations or accounting processes are not included.
- Detailed Case Documents
  - Storing detailed legal documents related to cases are not part of this database.

Data and Requirements Analysis:

- Employee Data: Captures essential employee details such as SSN, name, and role within the firm.
- Client Data: Stores client information, distinguishing between individuals and companies, with their contact details.
- Case Data: Manages details related to cases, their resolution status, relevancy periods, and associated clients and lawyers.
- Trial Data: Organizes trial information, associated with cases and representatives participating.
- Payment Data: Tracks payment details between clients and lawyers, ensuring proper resolution and tracking debt status.

Business Processes:

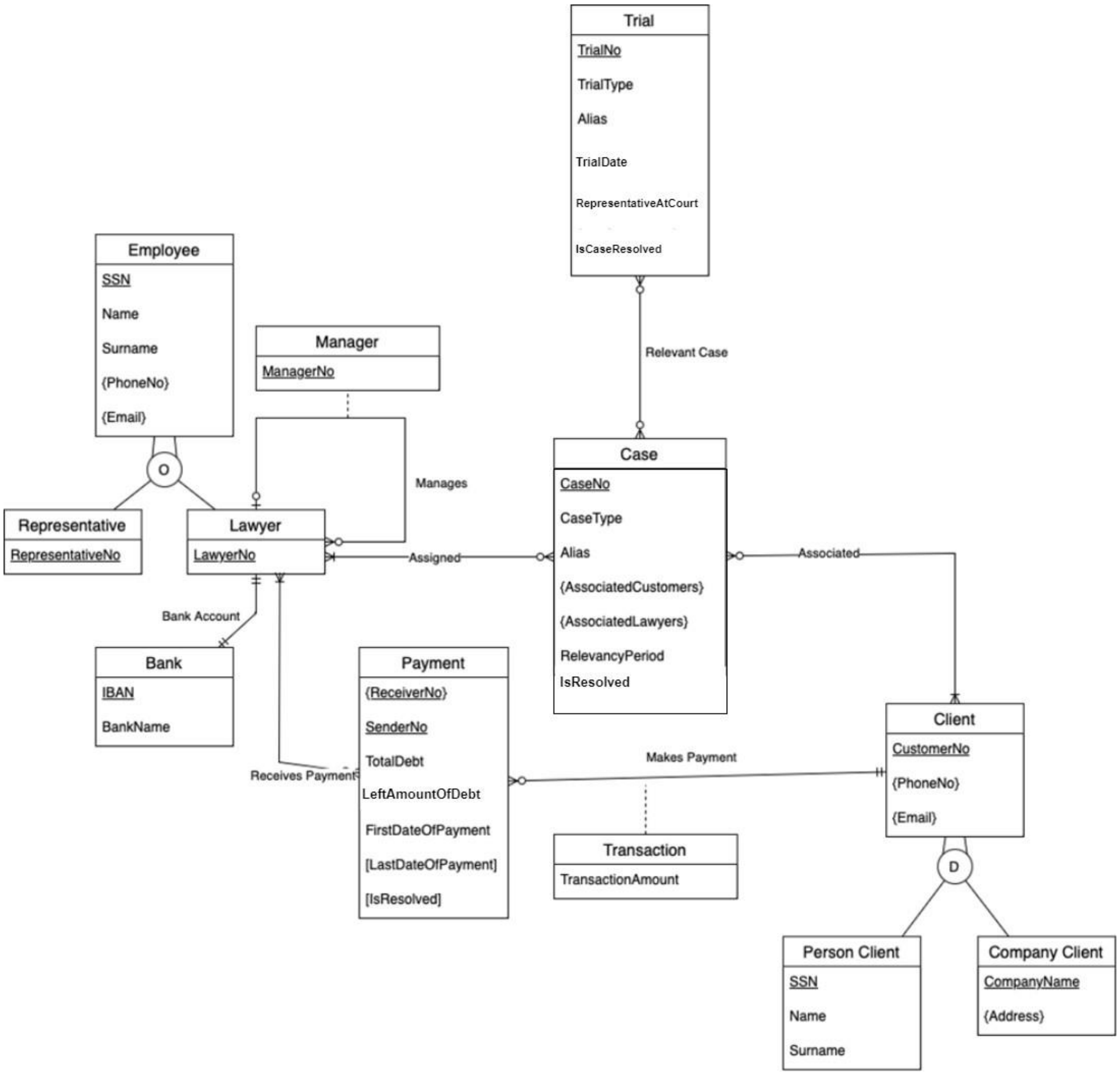
Supported Processes:

Case Management: Tracking case details, associating clients and lawyers.
Payment Tracking: Managing payments and tracking debts.
Employee and Client Management: Recording employee and client details, including their contact information.

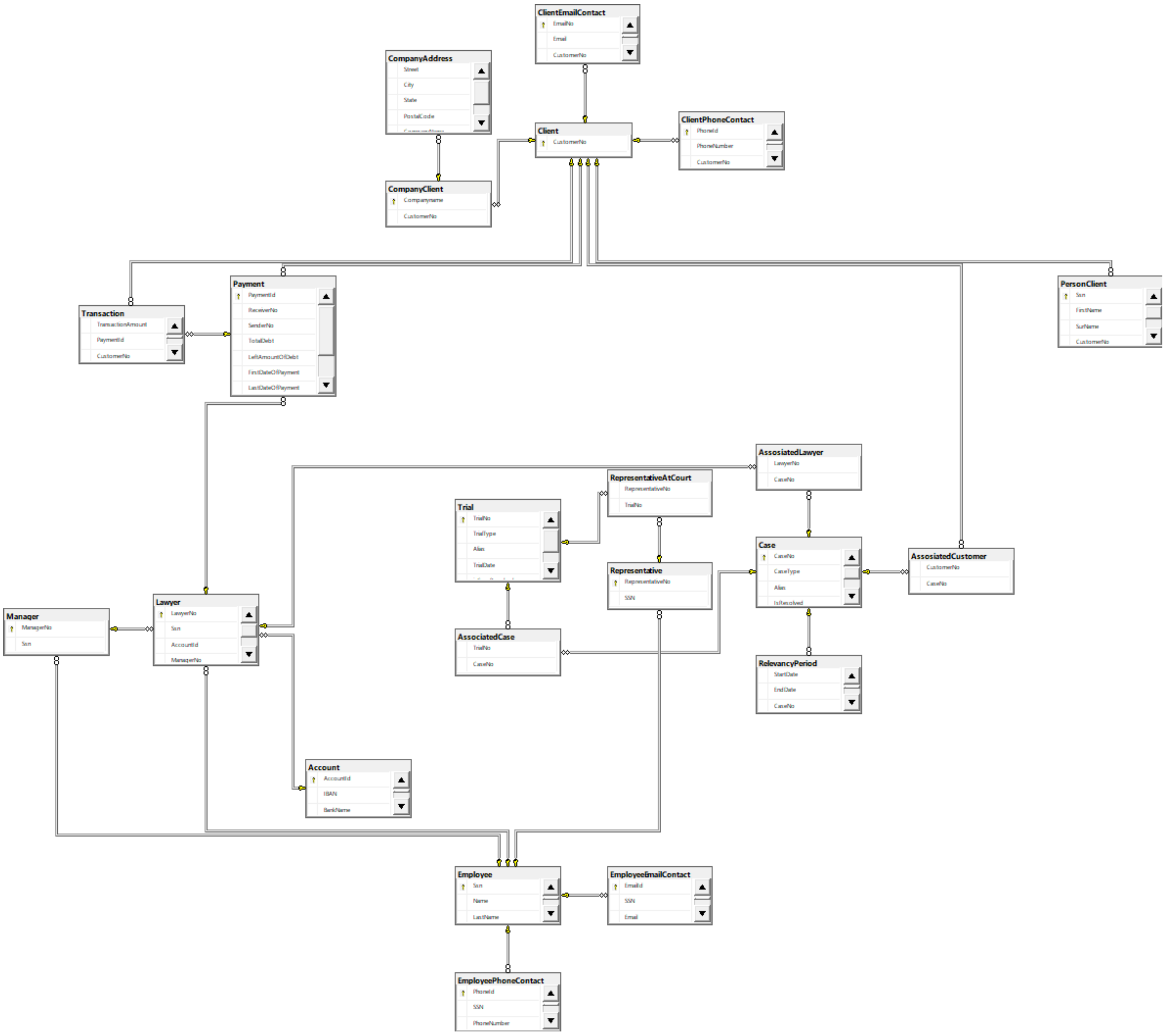
Not Supported Processes:

Legal Documentation Management: Detailed management of legal documents related to cases.
In-depth Financial Accounting: Complex financial calculations or comprehensive accounting processes beyond payment tracking.

ER Diagram


Trial
TrialNo
TrialType
Alias
TrialDate
RepresentativeAtCourt
IsCaseResolved
Relevant Case
Employee
SSN
Name
Surname
{PhoneNo}
{Email}
Manager
ManagerNo
Manages
O
Representative
RepresentativeNo
Lawyer
LawyerNo
Assigned
Case
CaseNo
CaseType
Alias
{AssociatedCustomers}
{AssociatedLawyers}
RelevancyPeriod
IsResolved
Associated
Bank Account
Bank
IBAN
BankName
Payment
{ReceiverNo}
SenderNo
TotalDebt
LeftAmountOfDebt
FirstDateOfPayment
[LastDateOfPayment]
[IsResolved]
Receives Payment
Makes Payment
Transaction
TransactionAmount
Client
CustomerNo
{PhoneNo}
{Email}
D
Person Client
SSN
Name
Surname
Company Client
CompanyName
{Address}

Database Diagram:



Tables:

Employee related tables:

- Employee: All employees working in this firm are stored in this table
  - ❖ Ssn NVARCHAR(12) PRIMARY KEY
  - ❖ Name NVARCHAR (50)
  - ❖ LastName NVARCHAR (50)

- EmployeePhoneContact: Phone numbers of employees
  - ❖ PhoneId int PRIMARY KEY
  - ❖ PhoneNumber NVARCHAR(20)
  - ❖ Ssn NVARCHAR(12)
    - i) This SSN is a Foreign Key referring to Employee

- EmployeeEmailContact: Emails of employees
  - ❖ EmailId int PRIMARY KEY
  - ❖ Email NVARCHAR(20)
  - ❖ Ssn NVARCHAR(12)
    - i) This SSN is a Foreign Key referring to Employee

- Representative: Representatves are lawyers attending to trials
  - ❖ RepresentativeNo int PRIMARY KEY
    - i) RepresentativeNo is an Identity starting from 200 and incrementing by 1
  - ❖ Ssn NVARCHAR(12)
    - i) This SSN is a Foreign Key referring to Employee

- Lawyer: Lawyers working in this firm
  - ❖ LawyerNo int PRIMARY KEY
    - i) LawyerNo is an Identity starting from 1 and incrementing by 1
  - ❖ Ssn NVARCHAR(12)
    - i) This SSN is a Foreign Key referring to Employee
  - ❖ AccountId int
    - i) This is a Foreign Key referring Account
  - ❖ ManagerNo int
    - i) This is a Foreign Key referring Manager

- Manager: Managers are lawyers who manages other lawyers
  - ❖ ManagerNo int PRIMARY KEY
    - i) ManagerNo is an Identity starting from 500 and incrementing by 1
  - ❖ Ssn NVARCHAR(12)
    - i) This SSN is a Foreign Key referring to Employee

Client related tables:

- Client: All customers are stored in this table
  - ❖ CustomerNo int PRIMARY KEY

- ClientPhoneContact: Phone numbers of clients are stored here
  - ❖ PhoneId NVARCHAR(10) PRIMARY KEY
  - ❖ PhoneNumber NVARCHAR(20)
  - ❖ CustomerNo int
    - i) This CustomerNo is Foreign Key referring to Client

- ClientEmailContact: Emails of clients are stored here
  - ❖ EmailNo int PRIMARY KEY
  - ❖ Email NVARCHAR(20)
  - ❖ CustomerNo int
    - i) This CustomerNo is Foreign Key referring to Client

- PersonClient: Individuils are stored in this table
  - ❖ Ssn NVARCHAR(12) Primary KEY
  - ❖ FirstName NVARCHAR(20)
  - ❖ SurName NVARCHAR(30)
  - ❖ CustomerNo int
    - i) This CustomerNo is Foreign Key referring to Client

- CompanyClient: Company clients are stored in this table
  - ❖ Companyname NVARCHAR(50) PRIMARY KEY
  - ❖ CustomerNo int
    - i) This CustomerNo is Foreign Key referring to Client

- CompanyAddress: Address of companies
  - ❖ Street VARCHAR(100)
  - ❖ City VARCHAR(50)
  - ❖ State VARCHAR(50)
  - ❖ PostalCode VARCHAR(20)
  - ❖ CompanyName NVARCHAR(50)
    - i) This CompanyName is Foreign Key referring to CompanyClient

Case related tables:

- Case: Cases about clients are stored in this table
  - ❖ CaseNo int PRIMARY KEY
  - ❖ CaseType NVARCHAR(30)
  - ❖ Alias NVARCHAR(30)
  - ❖ IsResolved bit
    - i) This bit holds if case is resolved
    - ii) This value is 0 by DEFAULT
- RelevancyPeriod: This stores start and end date of case
  - ❖ StartDate DATE
  - ❖ EndDate DATE
  - ❖ CaseNo int
    - i) This CaseNo is Foreign Key referring to Case
- AssosiatedCustomer: This is association between case and client
  - ❖ CustomerNo int
    - i) This CustomerNo is Foreign Key referring to Client
  - ❖ CaseNo int
    - i) This CaseNo is Foreign Key referring to Case
- AssosiatedLawyer: This table stores lawyers assigned to this case
  - ❖ LawyerNo int
    - i) This LawyerNo is Foreign Key referring to Client
  - ❖ CaseNo int
    - i) This CaseNo is Foreign Key referring to Case

- Trial: Trials of cases are stored here, a trial may be about multiple cases
  - ❖ TrialNo int PRIMARY KEY
  - ❖ TrialType NVARCHAR(50)
  - ❖ Alias NVARCHAR(30)
  - ❖ TrialDate date
    - i) This is a Non Clustered Index
  - ❖ isCaseResolved BIT
    - i) If case of this trial is solved in this trial then this value will be 1
    - ii) This value is 0 by DEFAULT
- RepresentativeAtCourt:
  - ❖ RepresentativeNo int
    - i) Representative assigned to this trial
    - ii) This RepresentativeNo is Foreign Key referring to Representative
  - ❖ TrialNo int
    - i) This TrialNo is Foreign Key referring to Trial
- AssociatedCase: This is association between cases and trials
  - ❖ TrialNo int
    - i) This TrialNo is Foreign Key referring to Trial
  - ❖ CaseNo int
    - i) This CaseNo is Foreign Key referring to Case
    - ii) This is also a Clustered Index

Payment related tables:

- Payment: This table stores payments to lawyer of client, like total debt, left amount of debt.
  - ❖ PaymentId int PRIMARY KEY
  - ❖ ReceiverNo int
    1. i) This ReceiverNo is Foreign Key and refers to Lawyer
  - ❖ SenderNo int
    1. i) This SenderNo is Foreign Key and refers to Client
  - ❖ TotalDebt DECIMAL(10, 2)
    1. i) Total charge client owe to lawyer
  - ❖ LeftAmountOfDebt DECIMAL(10, 2)
    1. i) This is being updated by trigger t_InsertTransaction, and shows left amount of debt
  - ❖ FirstDateOfPayment DATE
    - i) This data is being constrained by condition below
    - ii) New payments should start in at most 6 months
    - iii) CHECK (FirstDateOfPayment <= DATEADD(MONTH, 6, GETDATE()))
    - iv) This is also a Non Clustered Index
  - ❖ LastDateOfPayment DATE
    - i) This stores the due date the payment shoul be paid of, which is 6 months later than first payment.
    - ii) This is a calculated data, calculated as below
    - iii) (DATEADD(MONTH, 6, FirstDateOfPayment))
    - iv) This is also a Non Clustered Index
  - ❖ IsResolved int
    - i) If all debt has been paid this value will become 1
    - ii) This is calculated by left amount of debt as below
    - iii) (CASE WHEN LeftAmountOfDebt <= 0 THEN 1 ELSE 0 END)

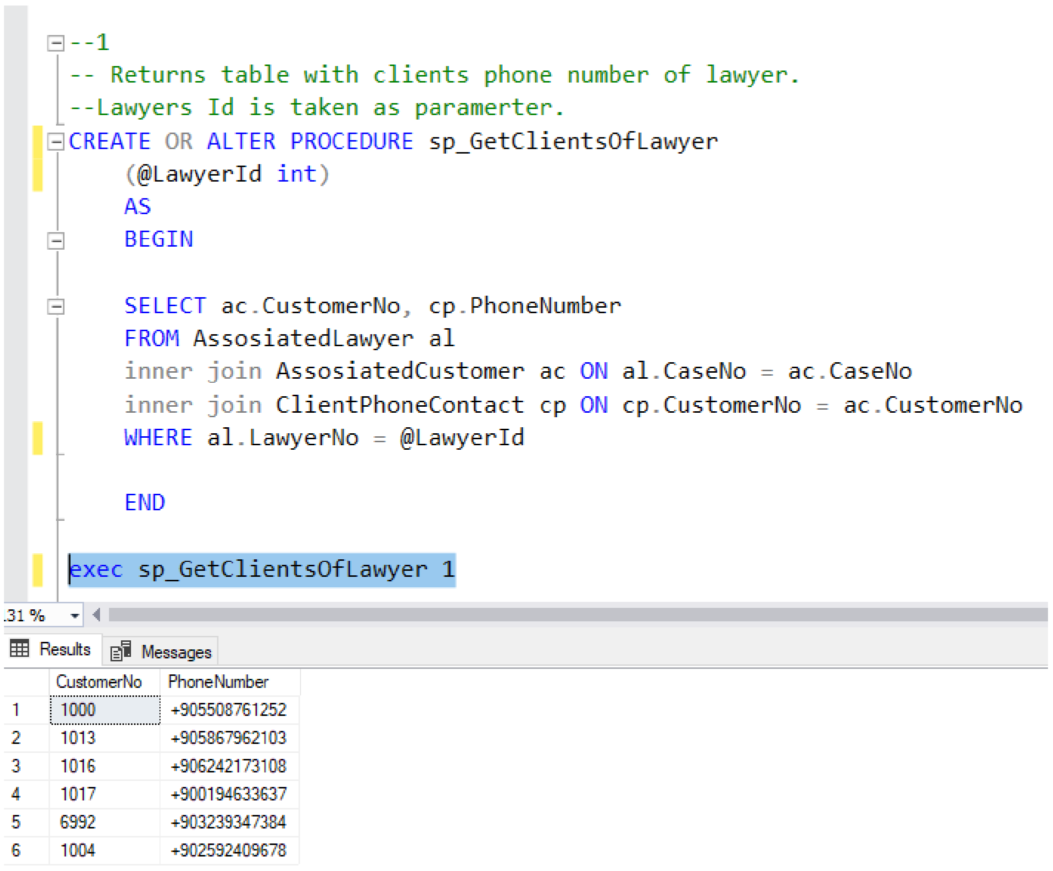
- Transaction: Stores every single payment made by client, also triggers t_InsertTransaction
  - ❖ TransactionAmount DECIMAL(10, 2)
  - ❖ PaymentId int
    - i) This PaymentId is a Foreign Key and refers to Payment
    - ii) This is also a Non Clustered Index
  - ❖ CustomerNo int
    - i) This CustomerNo is a Foreign Key and refers to Client

- Account: Account of lawyers. Stores informations like Iban and bank name
  - ❖ AccountId int PRIMARY KEY
  - ❖ IBAN NVARCHAR(30) UNIQUE
    - i) This is a Unique, every IBAN in system must be different
  - ❖ BankName NVARCHAR(20)

Stored Procedures:

1. sp_GetClientsOfLawyer

Returns table with clients phone number of lawyer. Lawyers Id is taken as parameter.

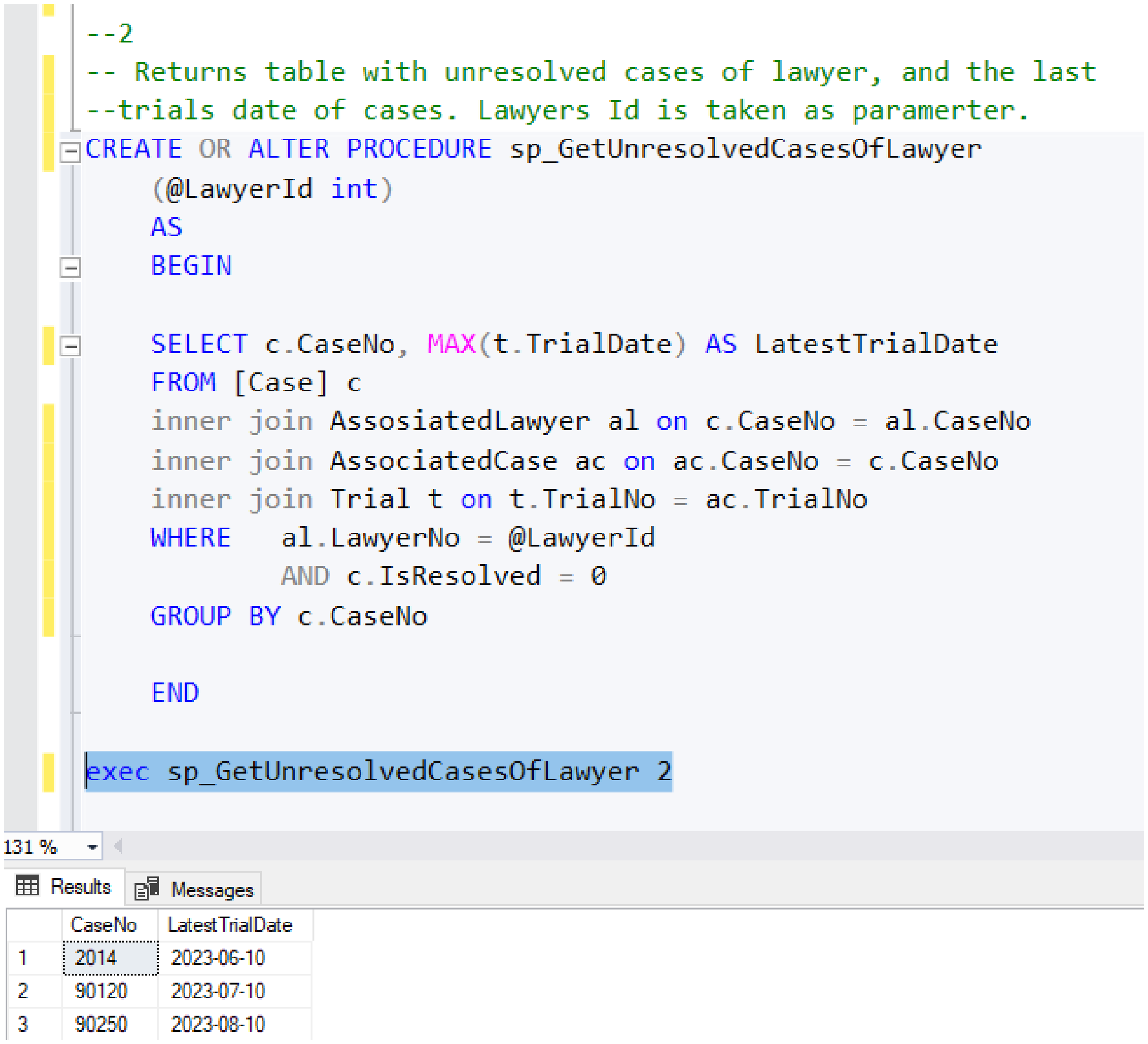


```
--1
-- Returns table with clients phone number of lawyer.
--Lawyers Id is taken as paramerter.
CREATE OR ALTER PROCEDURE sp_GetClientsOfLawyer
    (@LawyerId int)
    AS
    BEGIN

    SELECT ac.CustomerNo, cp.PhoneNumber
    FROM AssosiatedLawyer al
    inner join AssosiatedCustomer ac ON al.CaseNo = ac.CaseNo
    inner join ClientPhoneContact cp ON cp.CustomerNo = ac.CustomerNo
    WHERE al.LawyerNo = @LawyerId

    END

exec sp_GetClientsOfLawyer 1
```

	CustomerNo	PhoneNumber
1	1000	+905508761252
2	1013	+905867962103
3	1016	+906242173108
4	1017	+900194633637
5	6992	+903239347384
6	1004	+902592409678

2. sp_GetUnresolvedCasesOfLawyer

Returns table with unresolved cases of lawyer, and the last trials date of cases. Lawyers Id is taken as parameter.



```
--2
-- Returns table with unresolved cases of lawyer, and the last
--trials date of cases. Lawyers Id is taken as paramerter.
CREATE OR ALTER PROCEDURE sp_GetUnresolvedCasesOfLawyer
    (@LawyerId int)
    AS
    BEGIN

    SELECT c.CaseNo, MAX(t.TrialDate) AS LatestTrialDate
    FROM [Case] c
    inner join AssosiatedLawyer al on c.CaseNo = al.CaseNo
    inner join AssociatedCase ac on ac.CaseNo = c.CaseNo
    inner join Trial t on t.TrialNo = ac.TrialNo
    WHERE   al.LawyerNo = @LawyerId
            AND c.IsResolved = 0
    GROUP BY c.CaseNo

    END

exec sp_GetUnresolvedCasesOfLawyer 2
```

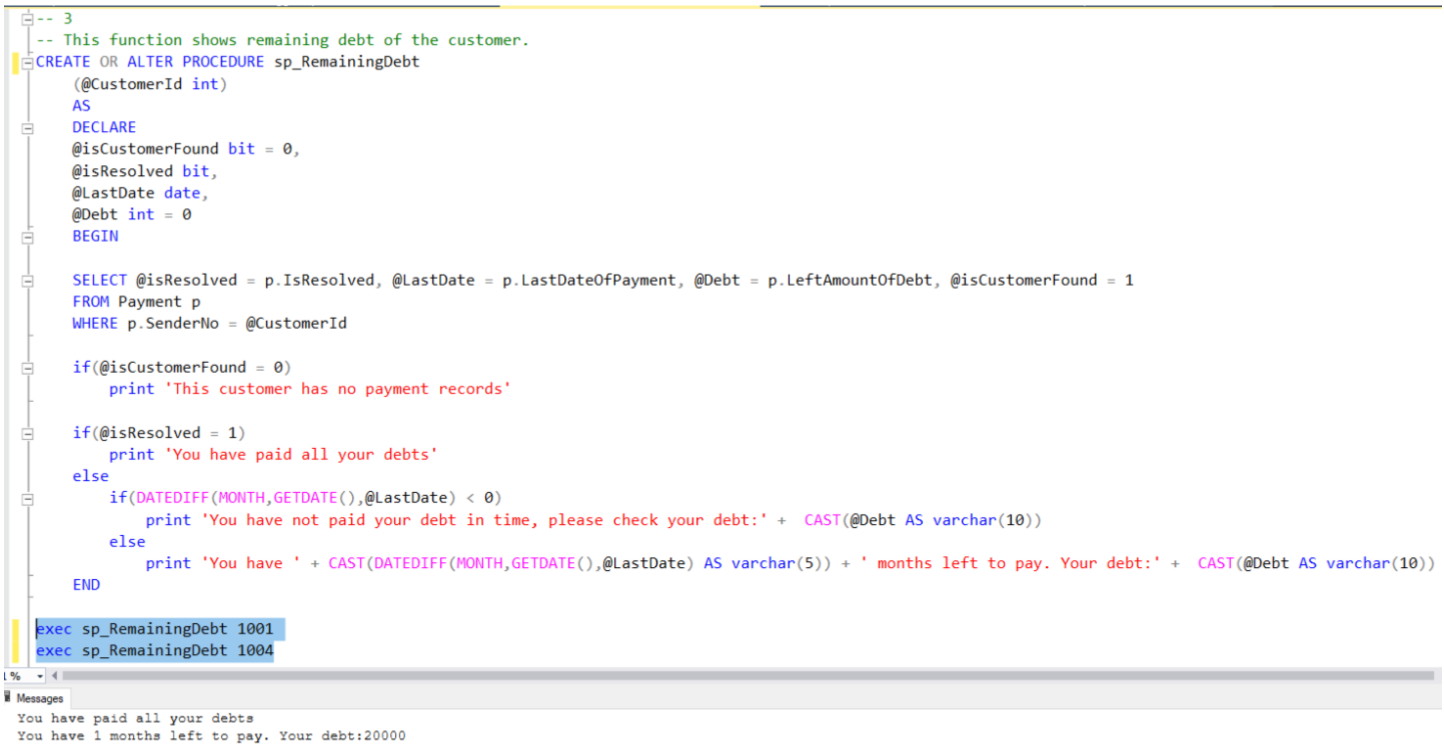
131 %

Results | Messages

	CaseNo	LatestTrialDate
1	2014	2023-06-10
2	90120	2023-07-10
3	90250	2023-08-10

3. sp_RemainingDebt

This function shows remaining debt of the customer and how many months they should pay in if they have any debt. Customer Id is taken as parameter.



```sql
-- 3
-- This function shows remaining debt of the customer.
CREATE OR ALTER PROCEDURE sp_RemainingDebt
    (@CustomerId int)
    AS
    DECLARE
    @isCustomerFound bit = 0,
    @isResolved bit,
    @LastDate date,
    @Debt int = 0
    BEGIN

    SELECT @isResolved = p.IsResolved, @LastDate = p.LastDateOfPayment, @Debt = p.LeftAmountOfDebt, @isCustomerFound = 1
    FROM Payment p
    WHERE p.SenderNo = @CustomerId

    if(@isCustomerFound = 0)
        print 'This customer has no payment records'

    if(@isResolved = 1)
        print 'You have paid all your debts'
    else
        if(DATEDIFF(MONTH,GETDATE(),@LastDate) < 0)
            print 'You have not paid your debt in time, please check your debt:' +  CAST(@Debt AS varchar(10))
        else
            print 'You have ' + CAST(DATEDIFF(MONTH,GETDATE(),@LastDate) AS varchar(5)) + ' months left to pay. Your debt:' +  CAST(@Debt AS varchar(10))
    END

exec sp_RemainingDebt 1001
exec sp_RemainingDebt 1004
```

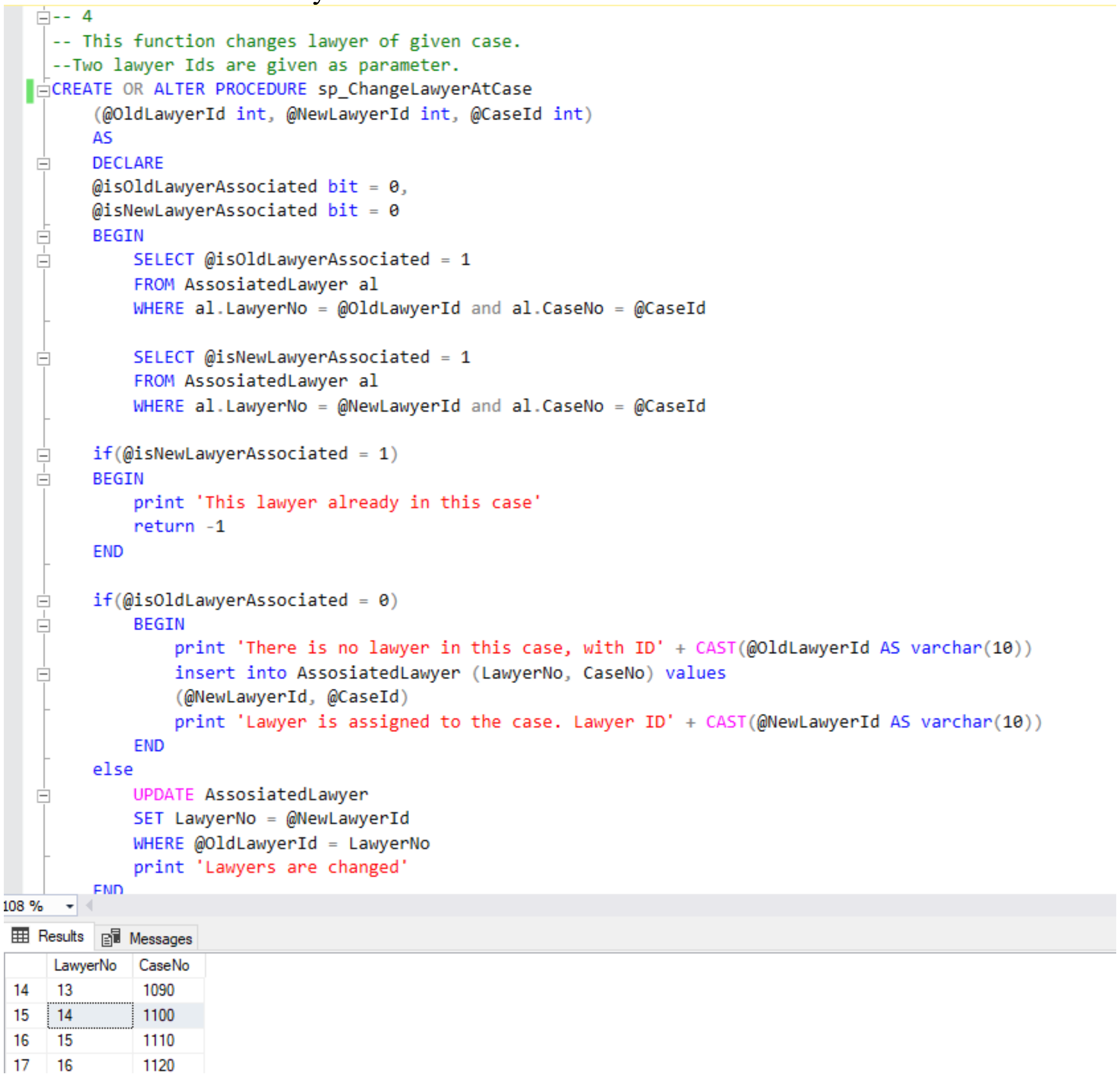
Messages

```
You have paid all your debts
You have 1 months left to pay. Your debt:20000
```

4. sp_ChangeLawyerAtCase

This function changes two lawyers on given case. Checks if lawyer is already in case or if lawyer wanted to be taken from case is in case or not. Two lawyer Ids and case no are given as parameter.

Code and AssociatedLawyer table before execution:



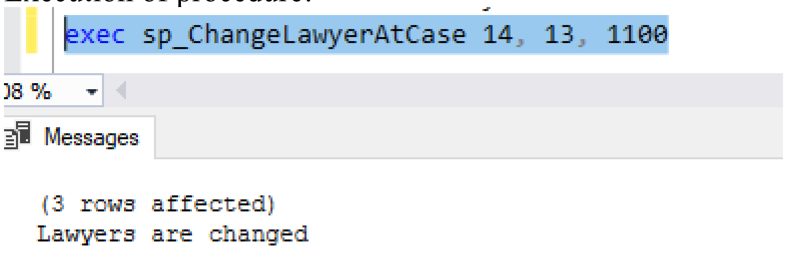
```
-- 4
-- This function changes lawyer of given case.
--Two lawyer Ids are given as parameter.
CREATE OR ALTER PROCEDURE sp_ChangeLawyerAtCase
    (@OldLawyerId int, @NewLawyerId int, @CaseId int)
    AS
    DECLARE
    @isOldLawyerAssociated bit = 0,
    @isNewLawyerAssociated bit = 0
    BEGIN
        SELECT @isOldLawyerAssociated = 1
        FROM AssosiatedLawyer al
        WHERE al.LawyerNo = @OldLawyerId and al.CaseNo = @CaseId

        SELECT @isNewLawyerAssociated = 1
        FROM AssosiatedLawyer al
        WHERE al.LawyerNo = @NewLawyerId and al.CaseNo = @CaseId

    if(@isNewLawyerAssociated = 1)
    BEGIN
        print 'This lawyer already in this case'
        return -1
    END

    if(@isOldLawyerAssociated = 0)
        BEGIN
            print 'There is no lawyer in this case, with ID' + CAST(@OldLawyerId AS varchar(10))
            insert into AssosiatedLawyer (LawyerNo, CaseNo) values
            (@NewLawyerId, @CaseId)
            print 'Lawyer is assigned to the case. Lawyer ID' + CAST(@NewLawyerId AS varchar(10))
        END
    else
        UPDATE AssosiatedLawyer
        SET LawyerNo = @NewLawyerId
        WHERE @OldLawyerId = LawyerNo
        print 'Lawyers are changed'
    END
```

108 %

Results | Messages

	LawyerNo	CaseNo
14	13	1090
15	14	1100
16	15	1110
17	16	1120

Execution of procedure:



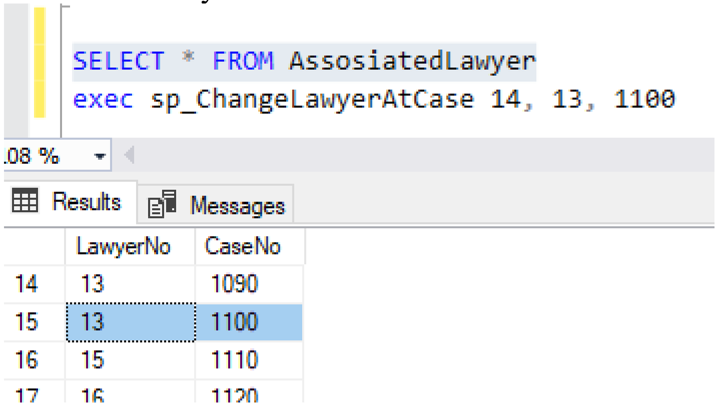
```
exec sp_ChangeLawyerAtCase 14, 13, 1100
```

08 %

Messages

```
(3 rows affected)
Lawyers are changed
```

AssociatedLawyer table after execution:

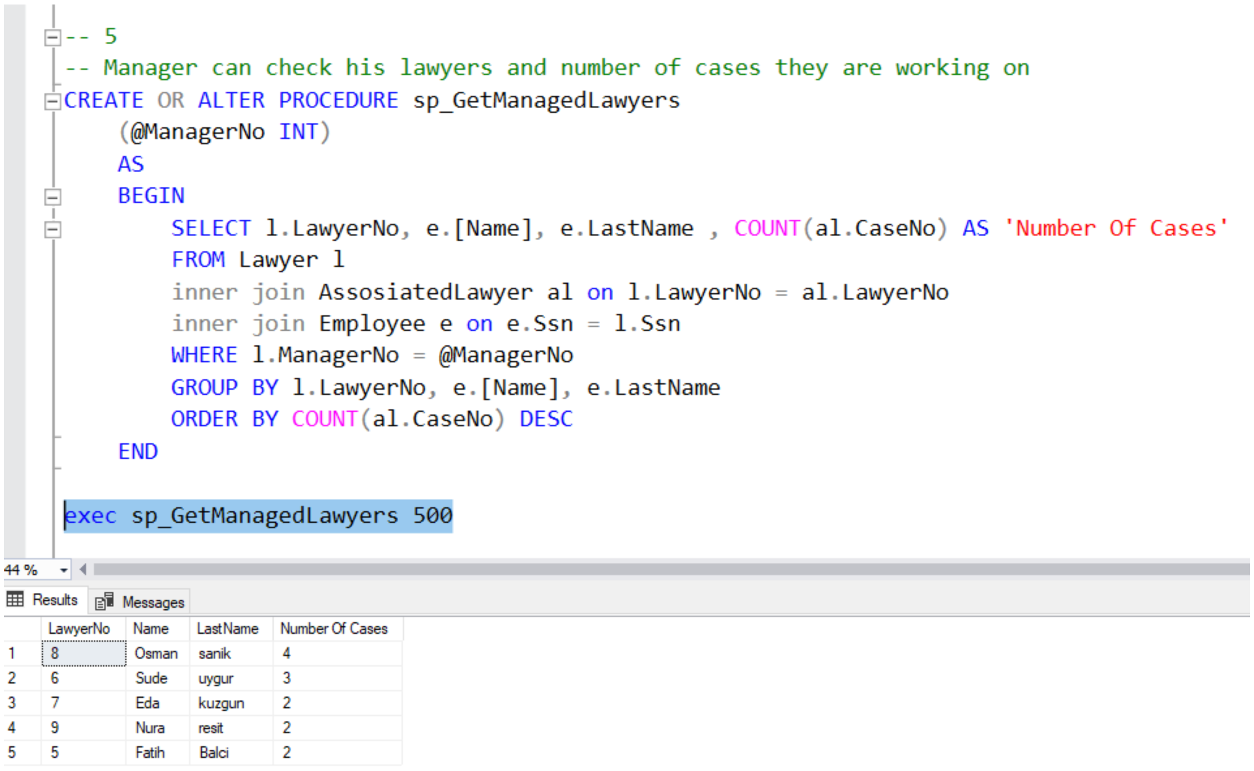


```
SELECT * FROM AssosiatedLawyer
exec sp_ChangeLawyerAtCase 14, 13, 1100
```

```
     LawyerNo   CaseNo
14   13         1090
15   13         1100
16   15         1110
17   16         1120
```

5. sp_GetManagedLawyers

Manager can check his lawyers and number of cases they are working on. ManagerNo is given as parameter.

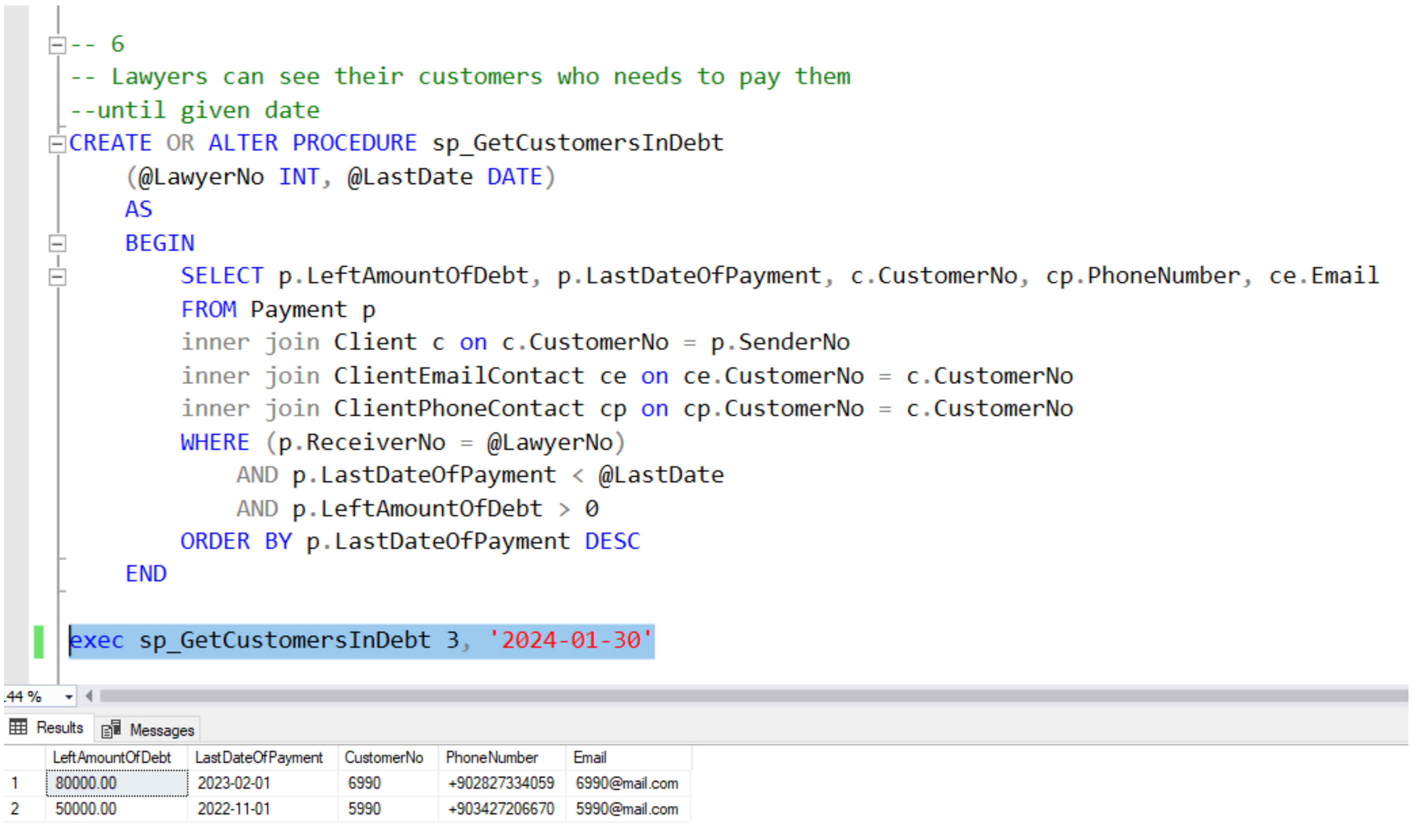


```
-- 5
-- Manager can check his lawyers and number of cases they are working on
CREATE OR ALTER PROCEDURE sp_GetManagedLawyers
    (@ManagerNo INT)
    AS
    BEGIN
        SELECT l.LawyerNo, e.[Name], e.LastName , COUNT(al.CaseNo) AS 'Number Of Cases'
        FROM Lawyer l
        inner join AssosiatedLawyer al on l.LawyerNo = al.LawyerNo
        inner join Employee e on e.Ssn = l.Ssn
        WHERE l.ManagerNo = @ManagerNo
        GROUP BY l.LawyerNo, e.[Name], e.LastName
        ORDER BY COUNT(al.CaseNo) DESC
    END

exec sp_GetManagedLawyers 500
```

```
    LawyerNo  Name   LastName  Number Of Cases
1   8         Osman  sanik     4
2   6         Sude   uygur     3
3   7         Eda    kuzgun    2
4   9         Nura   resit     2
5   5         Fatih  Balci     2
```

6. sp_GetCustomersInDebt

Lawyers can see their customers who needs to pay them until given date. LawyerNo and Date is taken as parameter.

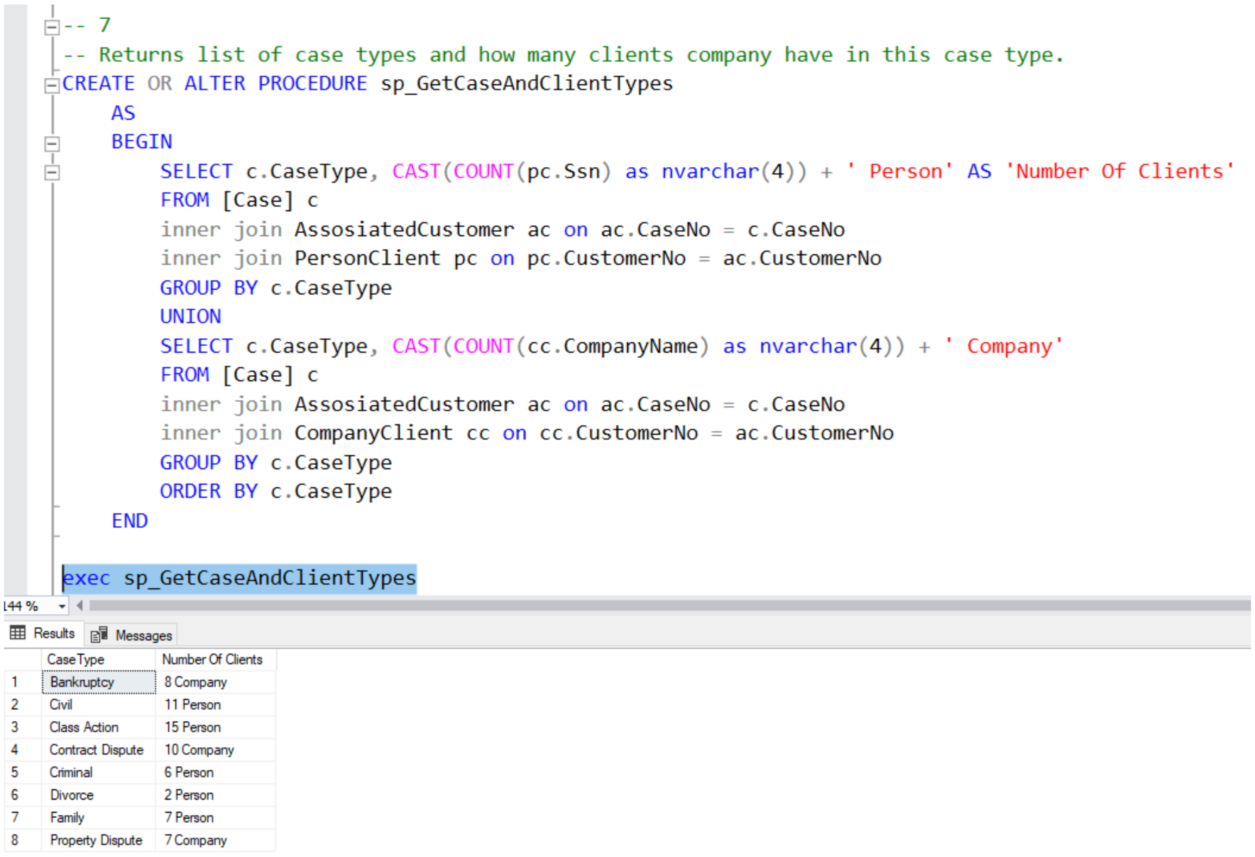


```
-- 6
-- Lawyers can see their customers who needs to pay them
--until given date
CREATE OR ALTER PROCEDURE sp_GetCustomersInDebt
    (@LawyerNo INT, @LastDate DATE)
    AS
    BEGIN
        SELECT p.LeftAmountOfDebt, p.LastDateOfPayment, c.CustomerNo, cp.PhoneNumber, ce.Email
        FROM Payment p
        inner join Client c on c.CustomerNo = p.SenderNo
        inner join ClientEmailContact ce on ce.CustomerNo = c.CustomerNo
        inner join ClientPhoneContact cp on cp.CustomerNo = c.CustomerNo
        WHERE (p.ReceiverNo = @LawyerNo)
            AND p.LastDateOfPayment < @LastDate
            AND p.LeftAmountOfDebt > 0
        ORDER BY p.LastDateOfPayment DESC
    END

exec sp_GetCustomersInDebt 3, '2024-01-30'
```

	LeftAmountOfDebt	LastDateOfPayment	CustomerNo	PhoneNumber	Email
1	80000.00	2023-02-01	6990	+902827334059	6990@mail.com
2	50000.00	2022-11-01	5990	+903427206670	5990@mail.com

7. sp_GetCaseAndClientTypes

Returns list of case types and how many clients company have in this case type, also type of customers if they are individuals or companies.



```
-- 7
-- Returns list of case types and how many clients company have in this case type.
CREATE OR ALTER PROCEDURE sp_GetCaseAndClientTypes
    AS
    BEGIN
        SELECT c.CaseType, CAST(COUNT(pc.Ssn) as nvarchar(4)) + ' Person' AS 'Number Of Clients'
        FROM [Case] c
        inner join AssosiatedCustomer ac on ac.CaseNo = c.CaseNo
        inner join PersonClient pc on pc.CustomerNo = ac.CustomerNo
        GROUP BY c.CaseType
        UNION
        SELECT c.CaseType, CAST(COUNT(cc.CompanyName) as nvarchar(4)) + ' Company'
        FROM [Case] c
        inner join AssosiatedCustomer ac on ac.CaseNo = c.CaseNo
        inner join CompanyClient cc on cc.CustomerNo = ac.CustomerNo
        GROUP BY c.CaseType
        ORDER BY c.CaseType
    END

exec sp_GetCaseAndClientTypes
```

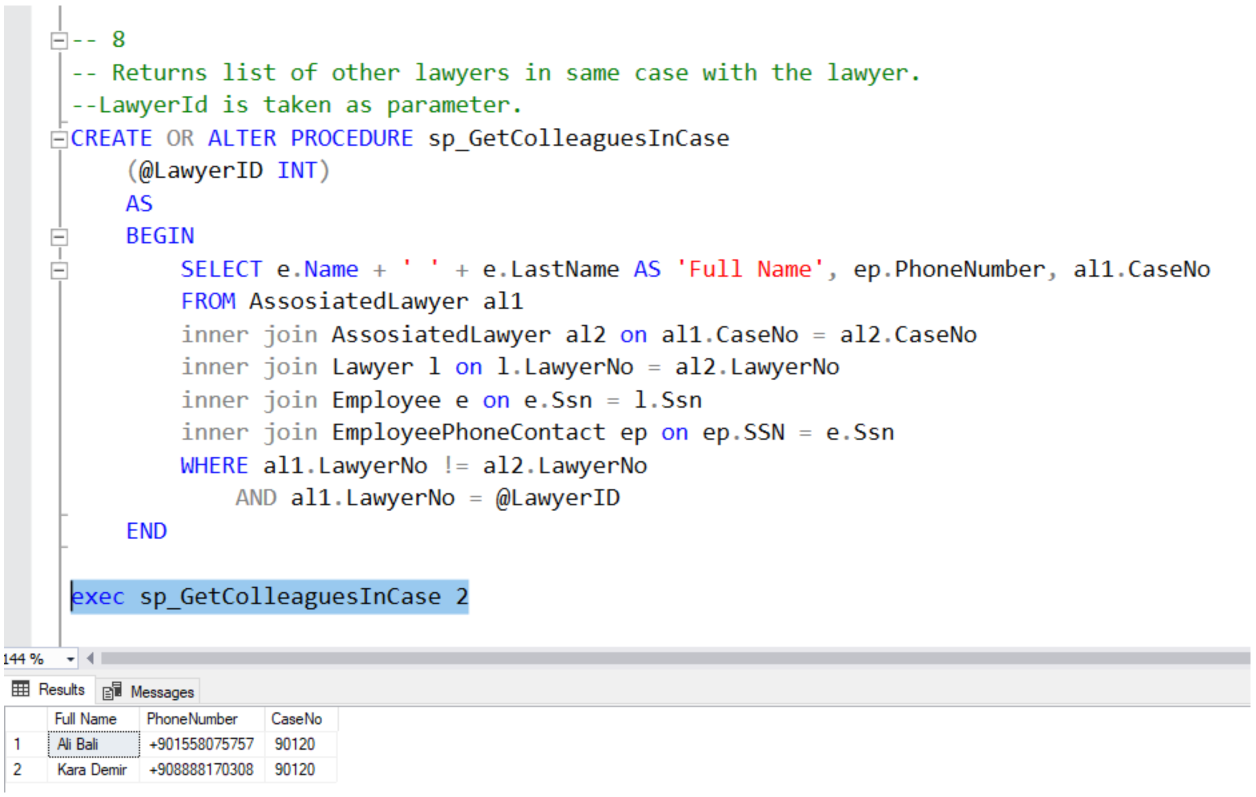
144 %

Results | Messages

	CaseType	Number Of Clients
1	Bankruptcy	8 Company
2	Civil	11 Person
3	Class Action	15 Person
4	Contract Dispute	10 Company
5	Criminal	6 Person
6	Divorce	2 Person
7	Family	7 Person
8	Property Dispute	7 Company

8. sp_GetColleaguesInCase

Returns a list of other attorneys in the same case as the lawyer. Lawyer ID is taken as a parameter.

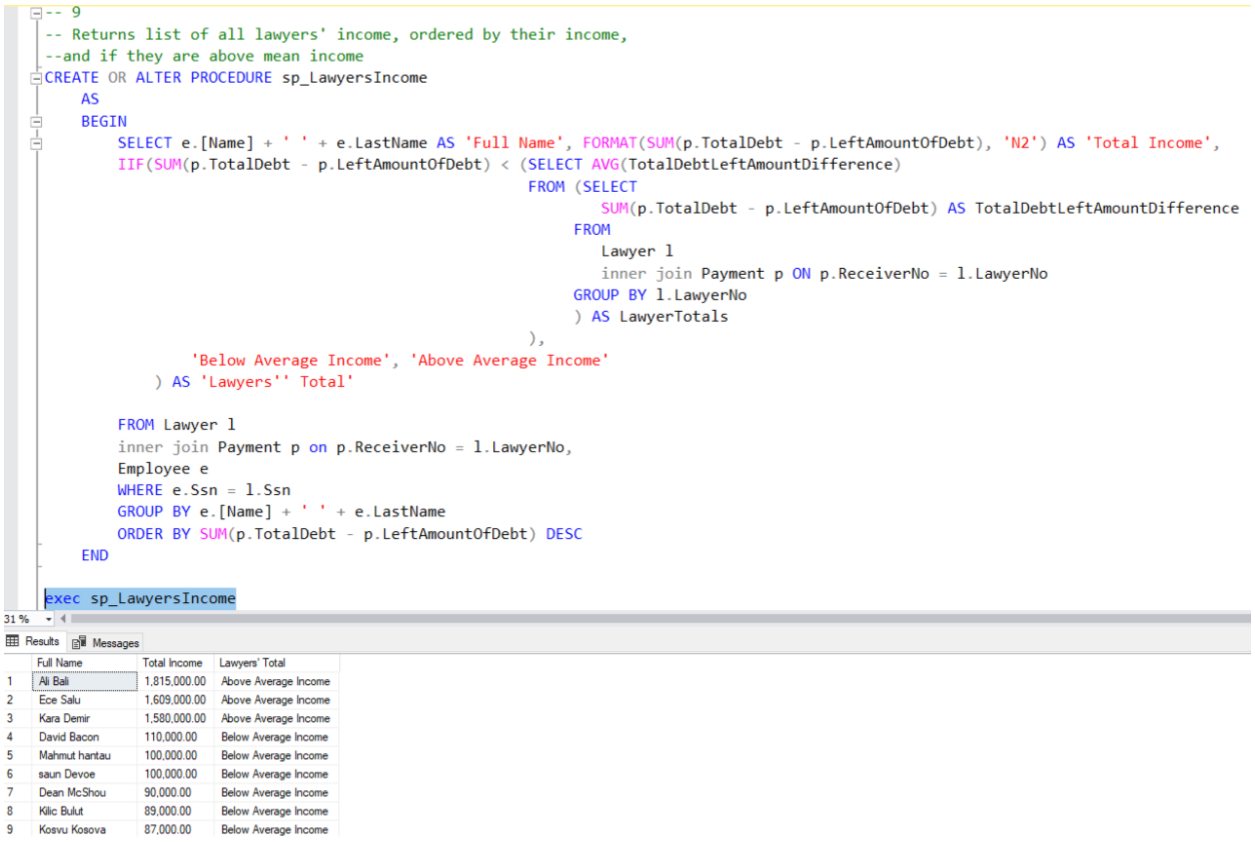


```
-- 8
-- Returns list of other lawyers in same case with the lawyer.
--LawyerId is taken as parameter.
CREATE OR ALTER PROCEDURE sp_GetColleaguesInCase
    (@LawyerID INT)
    AS
    BEGIN
        SELECT e.Name + ' ' + e.LastName AS 'Full Name', ep.PhoneNumber, al1.CaseNo
        FROM AssosiatedLawyer al1
        inner join AssosiatedLawyer al2 on al1.CaseNo = al2.CaseNo
        inner join Lawyer l on l.LawyerNo = al2.LawyerNo
        inner join Employee e on e.Ssn = l.Ssn
        inner join EmployeePhoneContact ep on ep.SSN = e.Ssn
        WHERE al1.LawyerNo != al2.LawyerNo
            AND al1.LawyerNo = @LawyerID
    END

exec sp_GetColleaguesInCase 2
```

	Full Name	PhoneNumber	CaseNo
1	Ali Bali	+901558075757	90120
2	Kara Demir	+908888170308	90120

9. sp_LawyersIncome

Returns list of all lawyers' income, ordered by their income, and if they are above mean income among all lawyers.



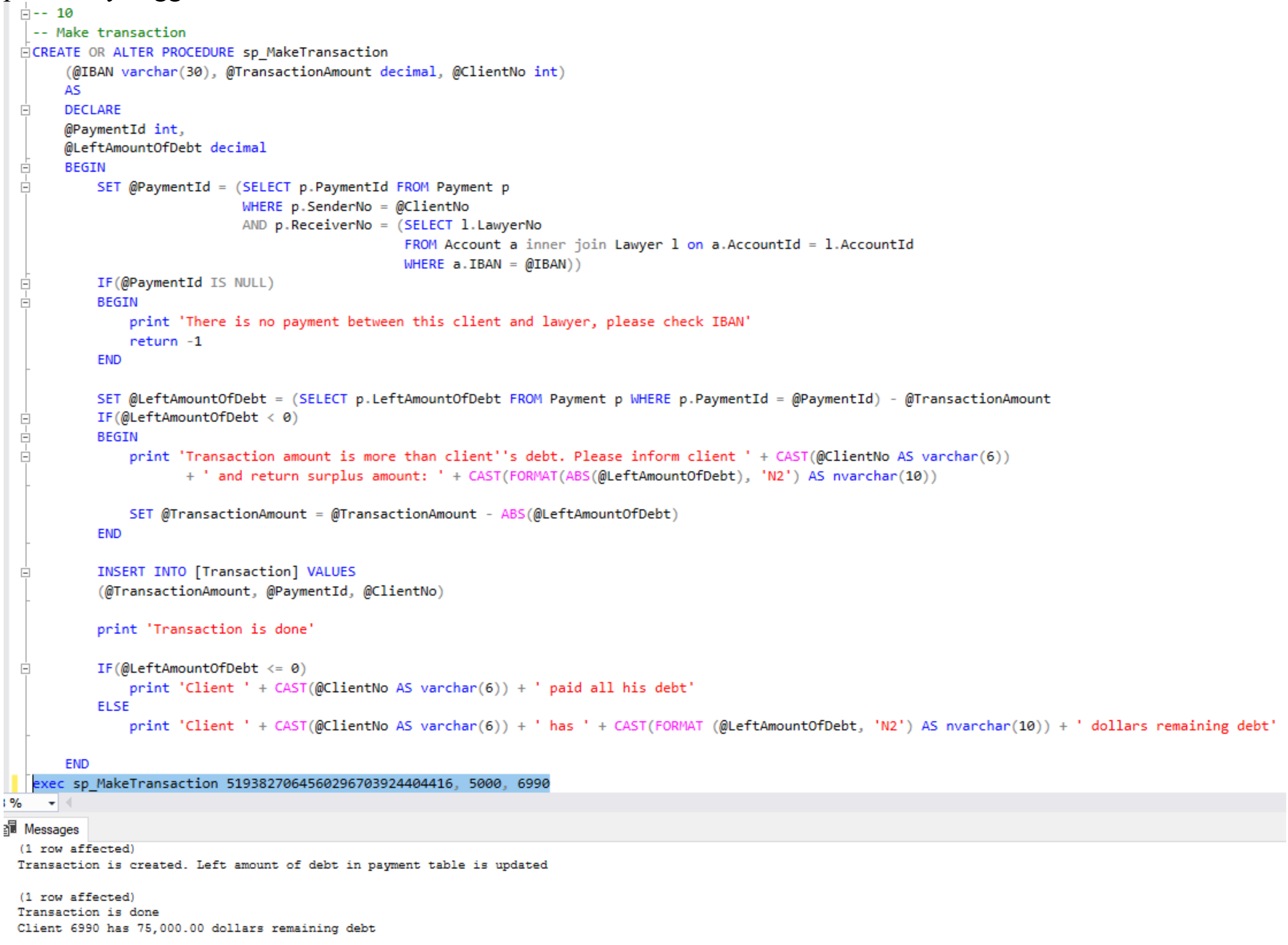
```sql
-- 9
-- Returns list of all lawyers' income, ordered by their income,
--and if they are above mean income
CREATE OR ALTER PROCEDURE sp_LawyersIncome
    AS
    BEGIN
        SELECT e.[Name] + ' ' + e.LastName AS 'Full Name', FORMAT(SUM(p.TotalDebt - p.LeftAmountOfDebt), 'N2') AS 'Total Income',
        IIF(SUM(p.TotalDebt - p.LeftAmountOfDebt) < (SELECT AVG(TotalDebtLeftAmountDifference)
                                                      FROM (SELECT
                                                            SUM(p.TotalDebt - p.LeftAmountOfDebt) AS TotalDebtLeftAmountDifference
                                                          FROM
                                                            Lawyer l
                                                            inner join Payment p ON p.ReceiverNo = l.LawyerNo
                                                          GROUP BY l.LawyerNo
                                                          ) AS LawyerTotals
                                                      ),
                'Below Average Income', 'Above Average Income'
            ) AS 'Lawyers'' Total'

        FROM Lawyer l
        inner join Payment p on p.ReceiverNo = l.LawyerNo,
        Employee e
        WHERE e.Ssn = l.Ssn
        GROUP BY e.[Name] + ' ' + e.LastName
        ORDER BY SUM(p.TotalDebt - p.LeftAmountOfDebt) DESC
    END

exec sp_LawyersIncome
```

	Full Name	Total Income	Lawyers' Total
1	Ali Bali	1,815,000.00	Above Average Income
2	Ece Salu	1,609,000.00	Above Average Income
3	Kara Demir	1,580,000.00	Above Average Income
4	David Bacon	110,000.00	Below Average Income
5	Mahmut hantau	100,000.00	Below Average Income
6	saun Devoe	100,000.00	Below Average Income
7	Dean McShou	90,000.00	Below Average Income
8	Kilic Bulut	89,000.00	Below Average Income
9	Kosvu Kosova	87,000.00	Below Average Income

10. sp_MakeTransaction

Create a new transaction. It checks if there is payment between lawyer and client if transaction amount is more than debt or not. IBAN, Transaction amount and client no is taken as parameter. Since this table inserts into transactions table, this will trigger t_InsertTransaction.

During execution, ‘Transaction is created. Left amount of debt in payment table is updated’ is printed by trigger:

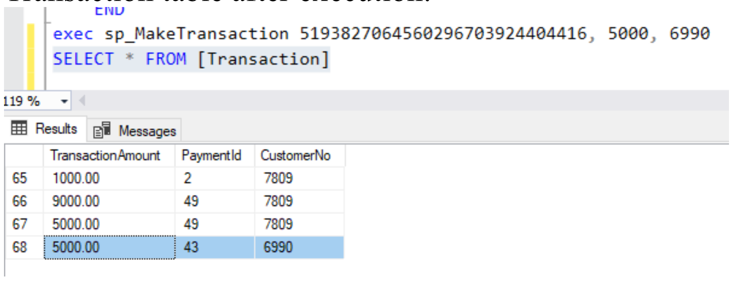


```
-- 10
-- Make transaction
CREATE OR ALTER PROCEDURE sp_MakeTransaction
    (@IBAN varchar(30), @TransactionAmount decimal, @ClientNo int)
    AS
    DECLARE
    @PaymentId int,
    @LeftAmountOfDebt decimal
    BEGIN
        SET @PaymentId = (SELECT p.PaymentId FROM Payment p
                          WHERE p.SenderNo = @ClientNo
                          AND p.ReceiverNo = (SELECT l.LawyerNo
                                              FROM Account a inner join Lawyer l on a.AccountId = l.AccountId
                                              WHERE a.IBAN = @IBAN))
        IF(@PaymentId IS NULL)
        BEGIN
            print 'There is no payment between this client and lawyer, please check IBAN'
            return -1
        END

        SET @LeftAmountOfDebt = (SELECT p.LeftAmountOfDebt FROM Payment p WHERE p.PaymentId = @PaymentId) - @TransactionAmount
        IF(@LeftAmountOfDebt < 0)
        BEGIN
            print 'Transaction amount is more than client''s debt. Please inform client ' + CAST(@ClientNo AS varchar(6))
                    + ' and return surplus amount: ' + CAST(FORMAT(ABS(@LeftAmountOfDebt), 'N2') AS nvarchar(10))

            SET @TransactionAmount = @TransactionAmount - ABS(@LeftAmountOfDebt)
        END

        INSERT INTO [Transaction] VALUES
        (@TransactionAmount, @PaymentId, @ClientNo)

        print 'Transaction is done'

        IF(@LeftAmountOfDebt <= 0)
            print 'Client ' + CAST(@ClientNo AS varchar(6)) + ' paid all his debt'
        ELSE
            print 'Client ' + CAST(@ClientNo AS varchar(6)) + ' has ' + CAST(FORMAT (@LeftAmountOfDebt, 'N2') AS nvarchar(10)) + ' dollars remaining debt'

    END
exec sp_MakeTransaction 5193827064560296703924404416, 5000, 6990
```

Messages

```
(1 row affected)
Transaction is created. Left amount of debt in payment table is updated

(1 row affected)
Transaction is done
Client 6990 has 75,000.00 dollars remaining debt
```

Transaction table after execution:



```
    END
exec sp_MakeTransaction 5193827064560296703924404416, 5000, 6990
SELECT * FROM [Transaction]
```

119 %

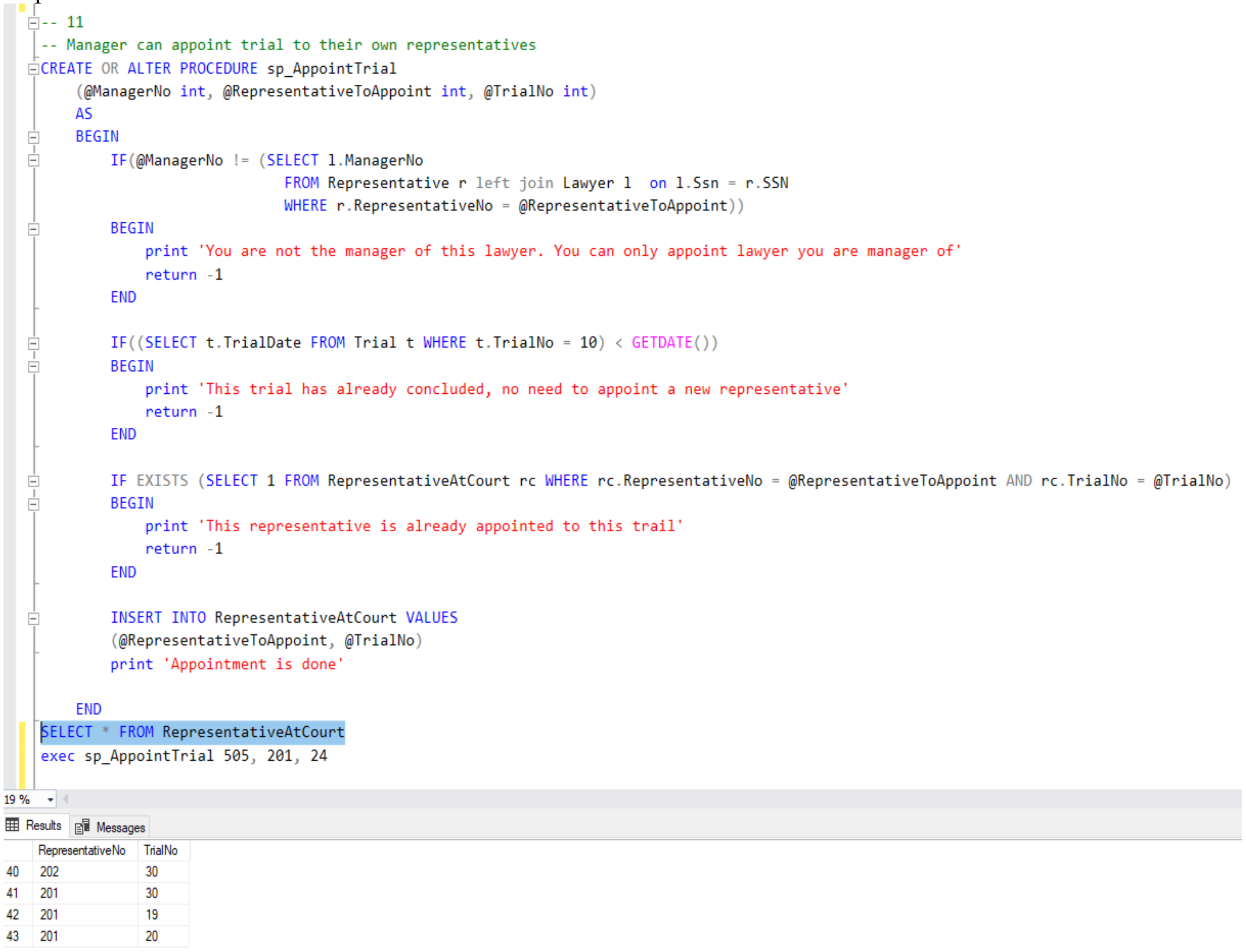
Results | Messages

	TransactionAmount	PaymentId	CustomerNo
65	1000.00	2	7809
66	9000.00	49	7809
67	5000.00	49	7809
68	5000.00	43	6990

11. sp_AppointTrial

Managers can appoint trial to their own representatives. Manager No, representative no and trial no is taken as parameters. It checks if representative is managed by this manager or not, or if this trial is already concluded, out dated.

RepresentativeAtCourt table before execution:



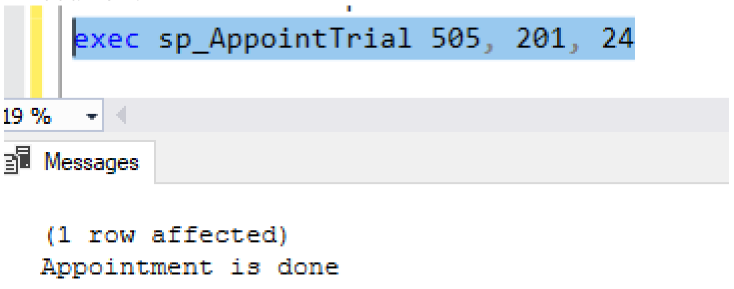
```sql
-- 11
-- Manager can appoint trial to their own representatives
CREATE OR ALTER PROCEDURE sp_AppointTrial
    (@ManagerNo int, @RepresentativeToAppoint int, @TrialNo int)
    AS
    BEGIN
        IF(@ManagerNo != (SELECT l.ManagerNo
                          FROM Representative r left join Lawyer l  on l.Ssn = r.SSN
                          WHERE r.RepresentativeNo = @RepresentativeToAppoint))
        BEGIN
            print 'You are not the manager of this lawyer. You can only appoint lawyer you are manager of'
            return -1
        END

        IF((SELECT t.TrialDate FROM Trial t WHERE t.TrialNo = 10) < GETDATE())
        BEGIN
            print 'This trial has already concluded, no need to appoint a new representative'
            return -1
        END

        IF EXISTS (SELECT 1 FROM RepresentativeAtCourt rc WHERE rc.RepresentativeNo = @RepresentativeToAppoint AND rc.TrialNo = @TrialNo)
        BEGIN
            print 'This representative is already appointed to this trail'
            return -1
        END

        INSERT INTO RepresentativeAtCourt VALUES
        (@RepresentativeToAppoint, @TrialNo)
        print 'Appointment is done'

    END
SELECT * FROM RepresentativeAtCourt
exec sp_AppointTrial 505, 201, 24
```

19 %

Results | Messages

	RepresentativeNo	TrialNo
40	202	30
41	201	30
42	201	19
43	201	20

Execution:



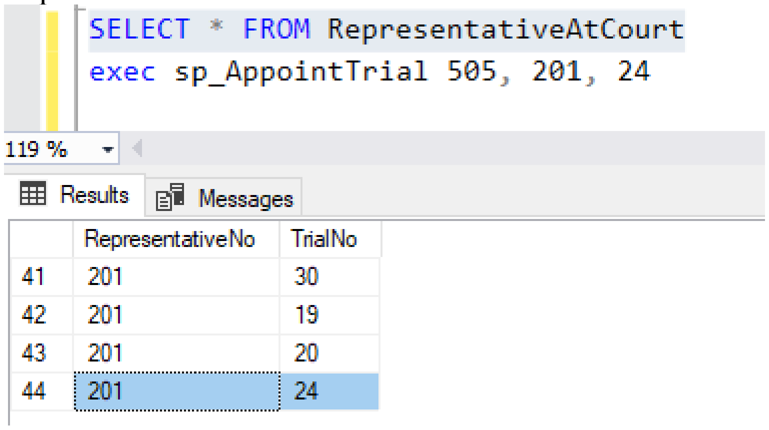
```
exec sp_AppointTrial 505, 201, 24
```

19 %

Messages

```
(1 row affected)
Appointment is done
```

RepresentativeAtCourt table after execution



```
SELECT * FROM RepresentativeAtCourt
exec sp_AppointTrial 505, 201, 24
```

119 %

Results | Messages

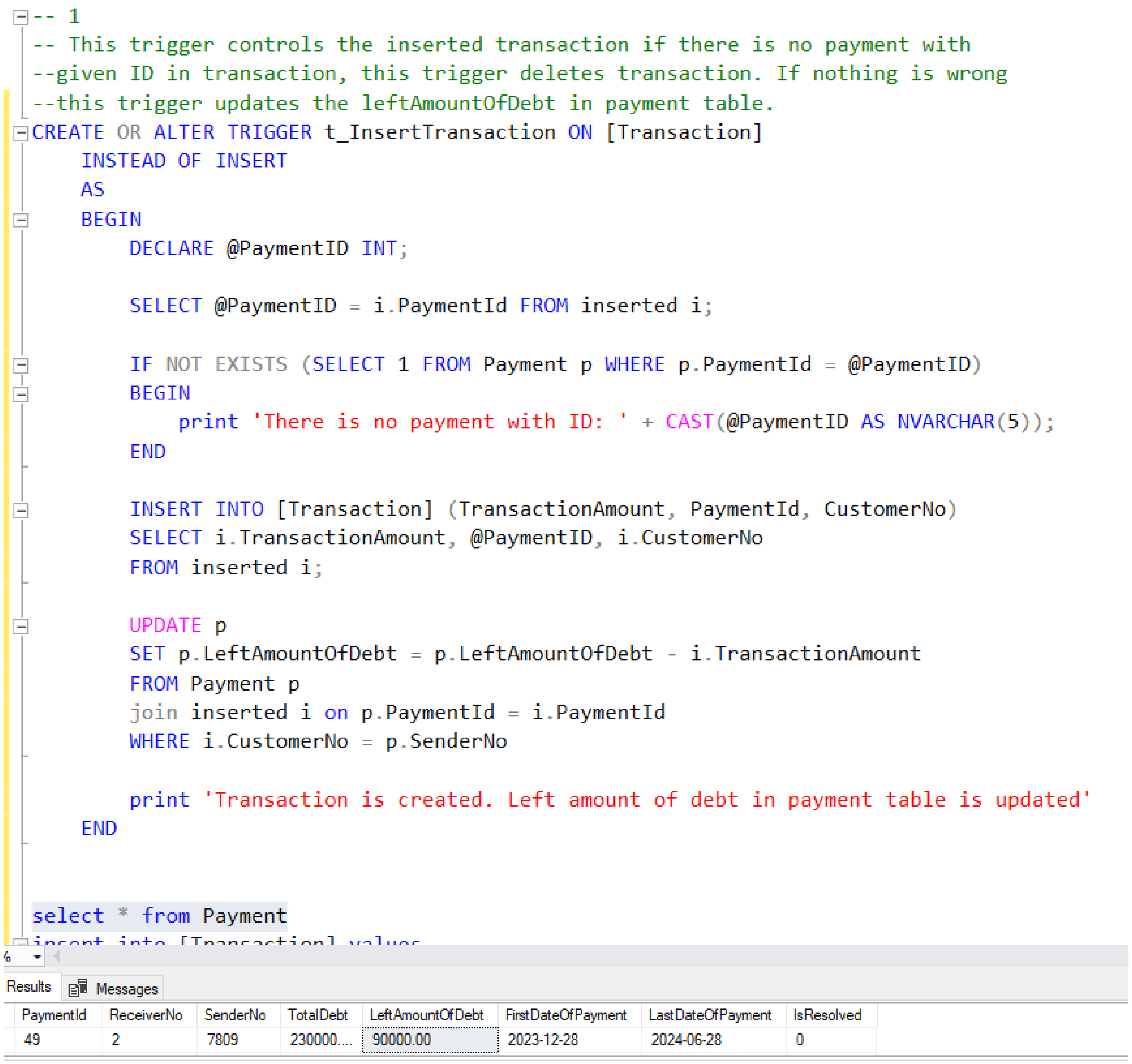
	RepresentativeNo	TrialNo
41	201	30
42	201	19
43	201	20
44	201	24

Triggers:

1. t_InsertTransaction

This trigger is triggered before insertion to transaction table. If there is no payment with the given ID in transaction, this trigger skips inserting transaction. If nothing is wrong this trigger updates the LeftAmountOfDebt in payment table of the client making transaction.

Payment table before insertion:

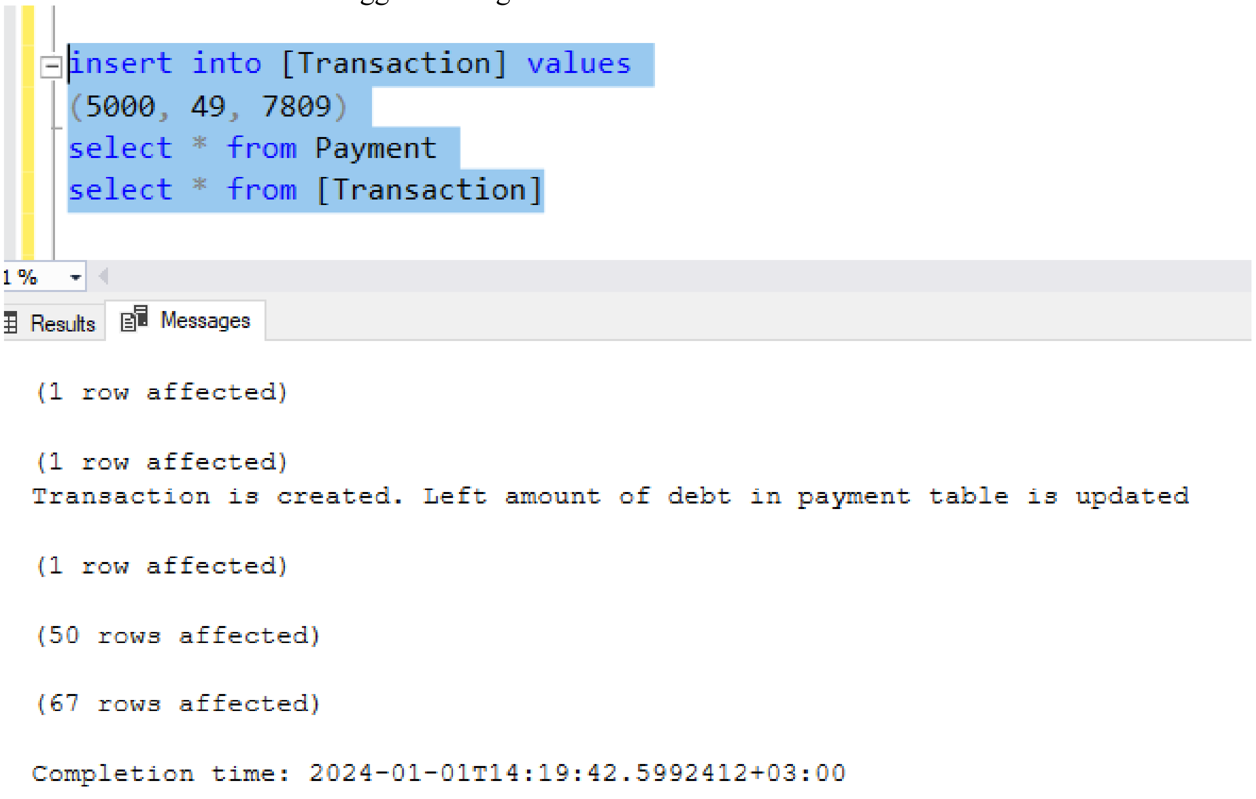


```
-- 1
-- This trigger controls the inserted transaction if there is no payment with
--given ID in transaction, this trigger deletes transaction. If nothing is wrong
--this trigger updates the leftAmountOfDebt in payment table.
CREATE OR ALTER TRIGGER t_InsertTransaction ON [Transaction]
    INSTEAD OF INSERT
    AS
    BEGIN
        DECLARE @PaymentID INT;

        SELECT @PaymentID = i.PaymentId FROM inserted i;

        IF NOT EXISTS (SELECT 1 FROM Payment p WHERE p.PaymentId = @PaymentID)
        BEGIN
            print 'There is no payment with ID: ' + CAST(@PaymentID AS NVARCHAR(5));
        END

        INSERT INTO [Transaction] (TransactionAmount, PaymentId, CustomerNo)
        SELECT i.TransactionAmount, @PaymentID, i.CustomerNo
        FROM inserted i;

        UPDATE p
        SET p.LeftAmountOfDebt = p.LeftAmountOfDebt - i.TransactionAmount
        FROM Payment p
        join inserted i on p.PaymentId = i.PaymentId
        WHERE i.CustomerNo = p.SenderNo

        print 'Transaction is created. Left amount of debt in payment table is updated'
    END


select * from Payment
insert into [Transaction] values
```

Results | Messages

	PaymentId	ReceiverNo	SenderNo	TotalDebt	LeftAmountOfDebt	FirstDateOfPayment	LastDateOfPayment	IsResolved
1	49	2	7809	230000....	90000.00	2023-12-28	2024-06-28	0

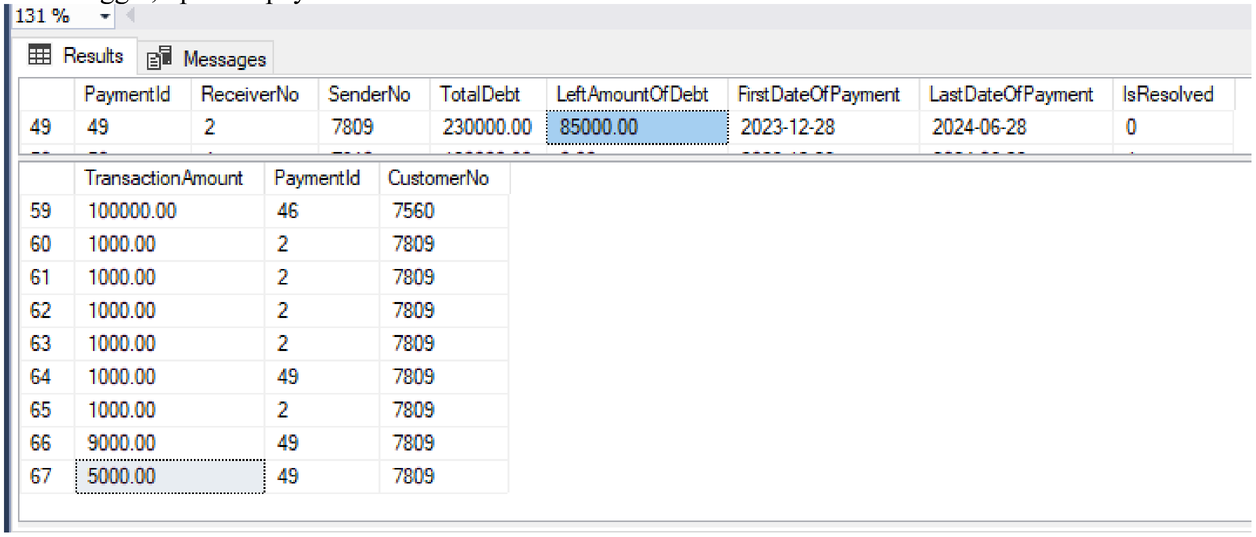
Insertion of transaction and trigger running:



```
insert into [Transaction] values
(5000, 49, 7809)
select * from Payment
select * from [Transaction]
```

```
(1 row affected)

(1 row affected)
Transaction is created. Left amount of debt in payment table is updated

(1 row affected)

(50 rows affected)

(67 rows affected)

Completion time: 2024-01-01T14:19:42.5992412+03:00
```

After trigger, updated payment and inserted transaction tables:



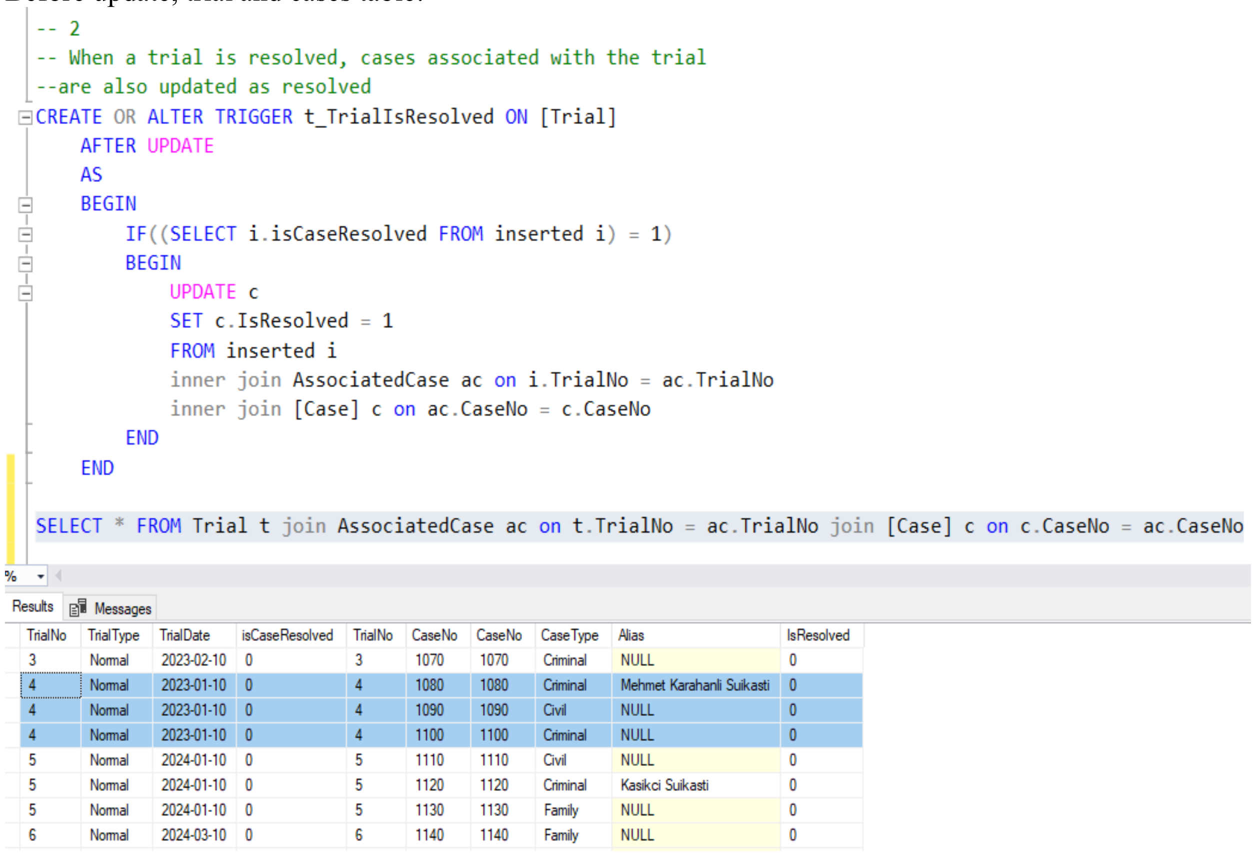
	PaymentId	ReceiverNo	SenderNo	TotalDebt	LeftAmountOfDebt	FirstDateOfPayment	LastDateOfPayment	IsResolved
49	49	2	7809	230000.00	85000.00	2023-12-28	2024-06-28	0

	TransactionAmount	PaymentId	CustomerNo
59	100000.00	46	7560
60	1000.00	2	7809
61	1000.00	2	7809
62	1000.00	2	7809
63	1000.00	2	7809
64	1000.00	49	7809
65	1000.00	2	7809
66	9000.00	49	7809
67	5000.00	49	7809

2. t_TrialIsResolved

When a trial is resolved, cases heard in this trial are updated as resolved. This is triggered by updating trials' isCaseResolved data.

Before update, trial and cases table:

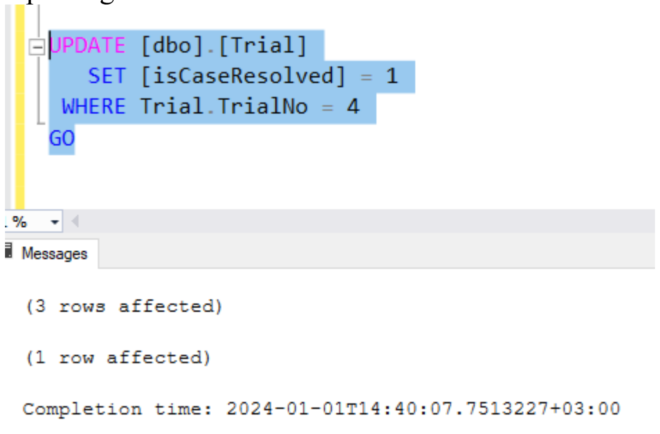


```
-- 2
-- When a trial is resolved, cases associated with the trial
--are also updated as resolved
CREATE OR ALTER TRIGGER t_TrialIsResolved ON [Trial]
    AFTER UPDATE
    AS
    BEGIN
        IF((SELECT i.isCaseResolved FROM inserted i) = 1)
        BEGIN
            UPDATE c
            SET c.IsResolved = 1
            FROM inserted i
            inner join AssociatedCase ac on i.TrialNo = ac.TrialNo
            inner join [Case] c on ac.CaseNo = c.CaseNo
        END
    END

SELECT * FROM Trial t join AssociatedCase ac on t.TrialNo = ac.TrialNo join [Case] c on c.CaseNo = ac.CaseNo
```

Results | Messages

TrialNo	TrialType	TrialDate	isCaseResolved	TrialNo	CaseNo	CaseNo	CaseType	Alias	IsResolved
3	Normal	2023-02-10	0	3	1070	1070	Criminal	NULL	0
4	Normal	2023-01-10	0	4	1080	1080	Criminal	Mehmet Karahanli Suikasti	0
4	Normal	2023-01-10	0	4	1090	1090	Civil	NULL	0
4	Normal	2023-01-10	0	4	1100	1100	Criminal	NULL	0
5	Normal	2024-01-10	0	5	1110	1110	Civil	NULL	0
5	Normal	2024-01-10	0	5	1120	1120	Criminal	Kasikci Suikasti	0
5	Normal	2024-01-10	0	5	1130	1130	Family	NULL	0
6	Normal	2024-03-10	0	6	1140	1140	Family	NULL	0

Updating trial isCaseResolved data:

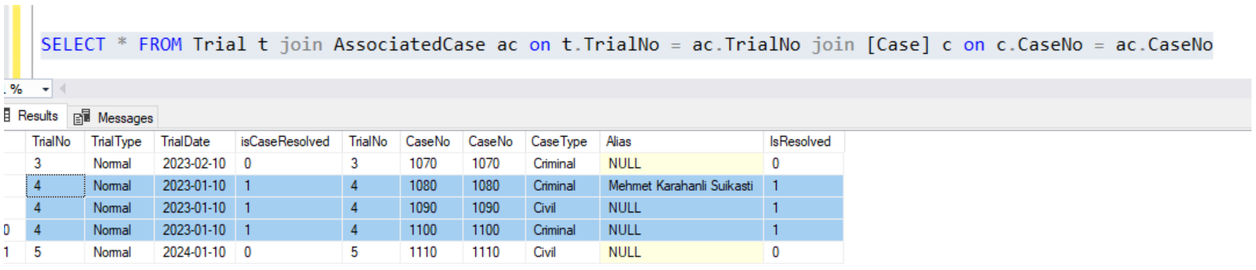


```
UPDATE [dbo].[Trial]
   SET [isCaseResolved] = 1
 WHERE Trial.TrialNo = 4
GO
```

Messages

```
(3 rows affected)

(1 row affected)

Completion time: 2024-01-01T14:40:07.7513227+03:00
```

After trigger, trial and case:



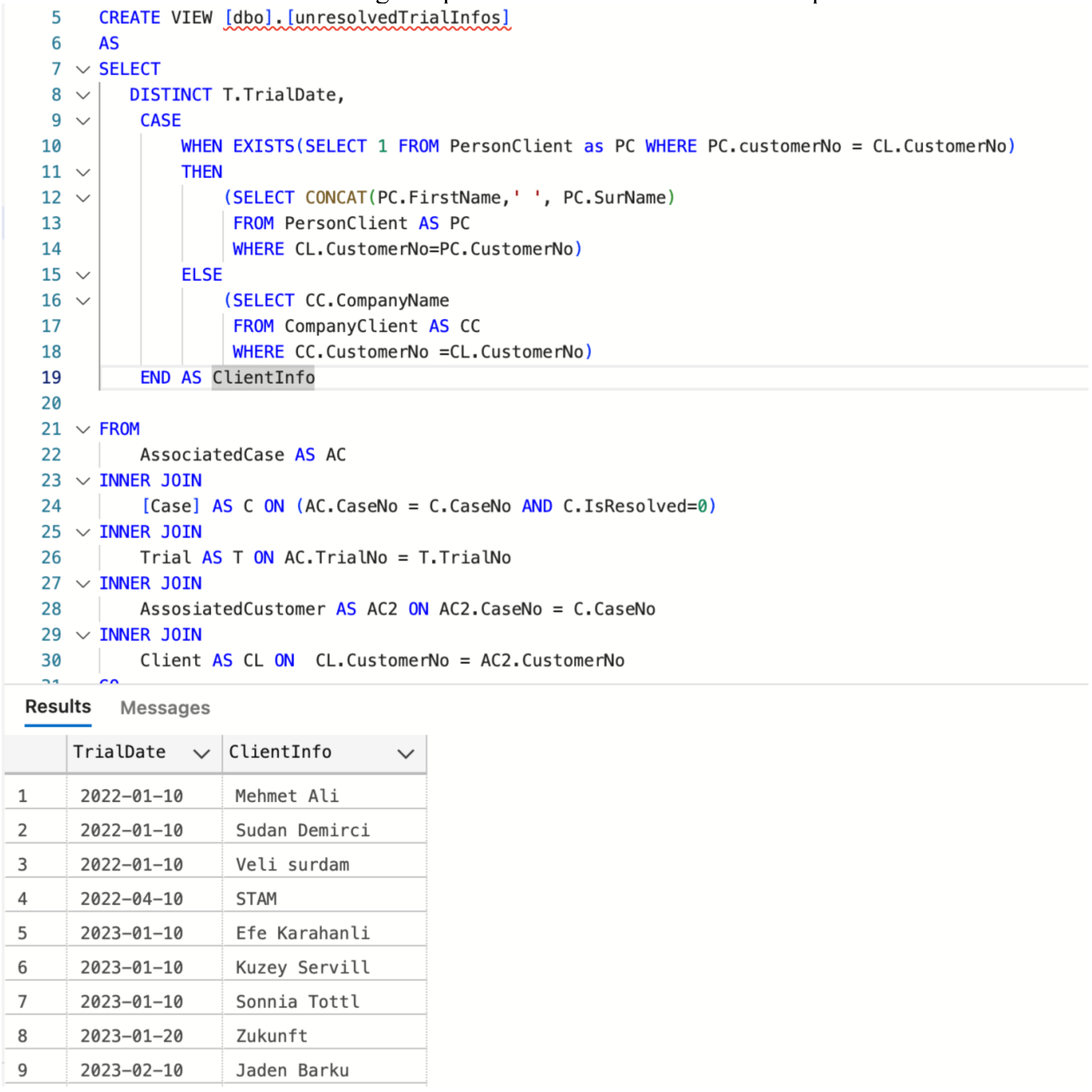
```
SELECT * FROM Trial t join AssociatedCase ac on t.TrialNo = ac.TrialNo join [Case] c on c.CaseNo = ac.CaseNo
```

	TrialNo	TrialType	TrialDate	isCaseResolved	TrialNo	CaseNo	CaseNo	CaseType	Alias	IsResolved
	3	Normal	2023-02-10	0	3	1070	1070	Criminal	NULL	0
	4	Normal	2023-01-10	1	4	1080	1080	Criminal	Mehmet Karahanli Suikasti	1
	4	Normal	2023-01-10	1	4	1090	1090	Civil	NULL	1
0	4	Normal	2023-01-10	1	4	1100	1100	Criminal	NULL	1
1	5	Normal	2024-01-10	0	5	1110	1110	Civil	NULL	0

Views:

1- unresolvedTrialInfos View

This view is used to see every unresolved trial date and the clients associated with the trial. Clients' info column shows distinguish person clients full name and companies' name.



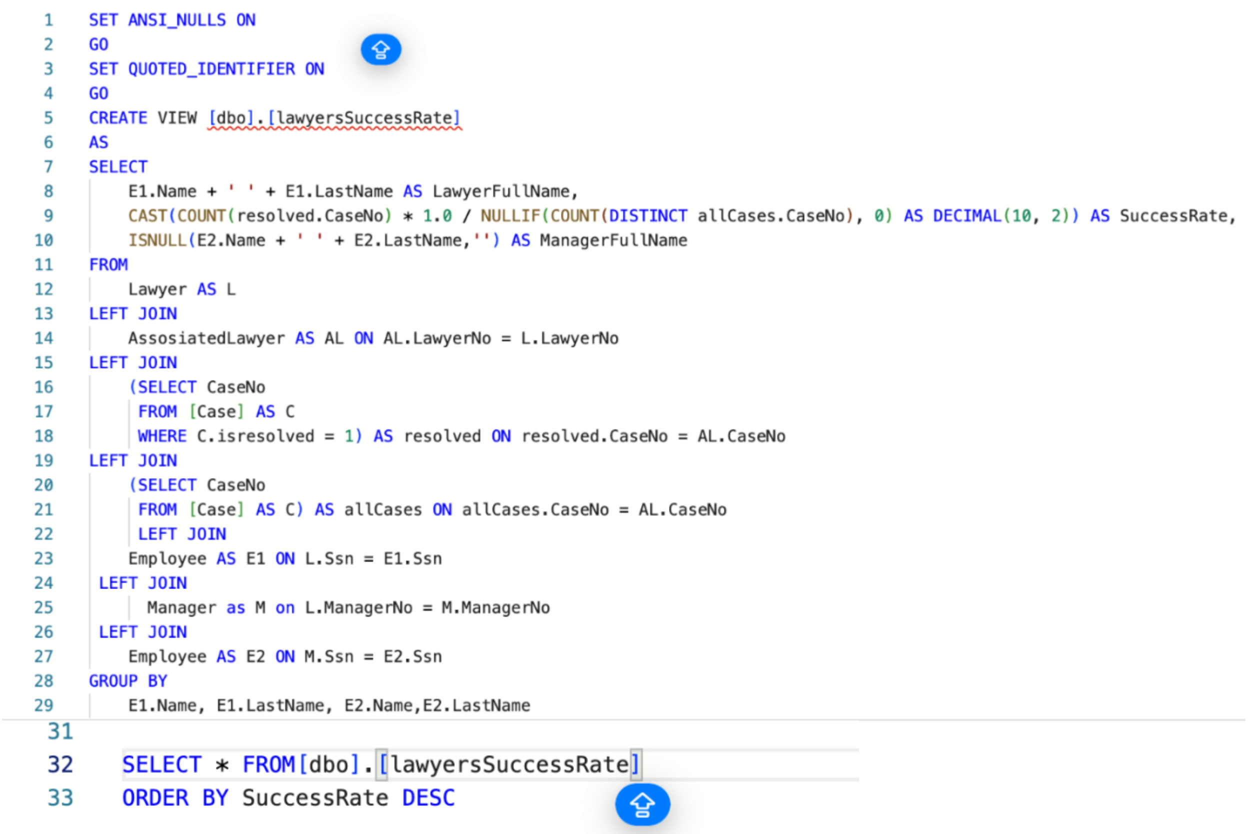
```
CREATE VIEW [dbo].[unresolvedTrialInfos]
AS
SELECT
   DISTINCT T.TrialDate,
    CASE
        WHEN EXISTS(SELECT 1 FROM PersonClient as PC WHERE PC.customerNo = CL.CustomerNo)
        THEN
            (SELECT CONCAT(PC.FirstName,' ', PC.SurName)
             FROM PersonClient AS PC
             WHERE CL.CustomerNo=PC.CustomerNo)
        ELSE
            (SELECT CC.CompanyName
             FROM CompanyClient AS CC
             WHERE CC.CustomerNo =CL.CustomerNo)
    END AS ClientInfo

FROM
    AssociatedCase AS AC
INNER JOIN
    [Case] AS C ON (AC.CaseNo = C.CaseNo AND C.IsResolved=0)
INNER JOIN
    Trial AS T ON AC.TrialNo = T.TrialNo
INNER JOIN
    AssosiatedCustomer AS AC2 ON AC2.CaseNo = C.CaseNo
INNER JOIN
    Client AS CL ON  CL.CustomerNo = AC2.CustomerNo
```

Results Messages

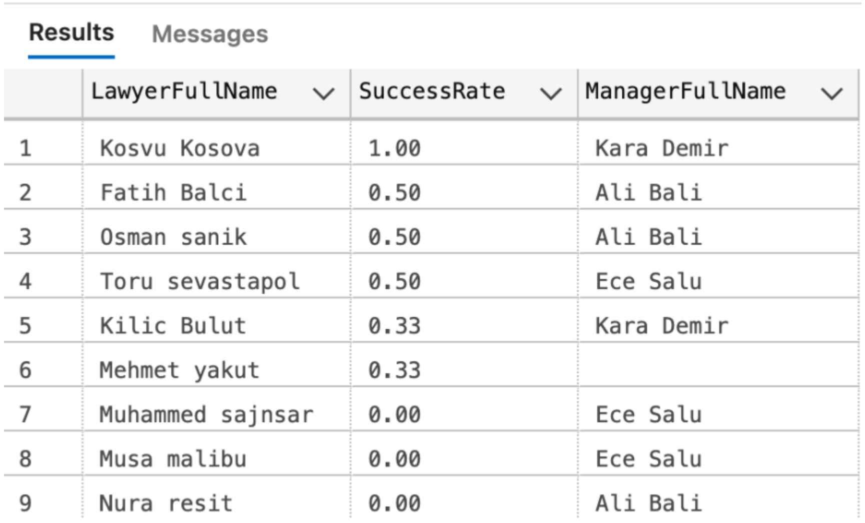
	TrialDate	ClientInfo
1	2022-01-10	Mehmet Ali
2	2022-01-10	Sudan Demirci
3	2022-01-10	Veli surdam
4	2022-04-10	STAM
5	2023-01-10	Efe Karahanli
6	2023-01-10	Kuzey Servill
7	2023-01-10	Sonnia Tottl
8	2023-01-20	Zukunft
9	2023-02-10	Jaden Barku

2- lawyersSuccessRate View

This view calculates the success rate, as a decimal, for each lawyer based on resolved cases compared to all cases they are associated with, showing the lawyer's full name and success rate and their managers.



```sql
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE VIEW [dbo].[lawyersSuccessRate]
AS
SELECT
    E1.Name + ' ' + E1.LastName AS LawyerFullName,
    CAST(COUNT(resolved.CaseNo) * 1.0 / NULLIF(COUNT(DISTINCT allCases.CaseNo), 0) AS DECIMAL(10, 2)) AS SuccessRate,
    ISNULL(E2.Name + ' ' + E2.LastName,'') AS ManagerFullName
FROM
    Lawyer AS L
LEFT JOIN
    AssosiatedLawyer AS AL ON AL.LawyerNo = L.LawyerNo
LEFT JOIN
    (SELECT CaseNo
     FROM [Case] AS C
     WHERE C.isresolved = 1) AS resolved ON resolved.CaseNo = AL.CaseNo
LEFT JOIN
    (SELECT CaseNo
     FROM [Case] AS C) AS allCases ON allCases.CaseNo = AL.CaseNo
     LEFT JOIN
    Employee AS E1 ON L.Ssn = E1.Ssn
 LEFT JOIN
      Manager as M on L.ManagerNo = M.ManagerNo
 LEFT JOIN
    Employee AS E2 ON M.Ssn = E2.Ssn
GROUP BY
    E1.Name, E1.LastName, E2.Name,E2.LastName

SELECT * FROM[dbo].[lawyersSuccessRate]
ORDER BY SuccessRate DESC
```

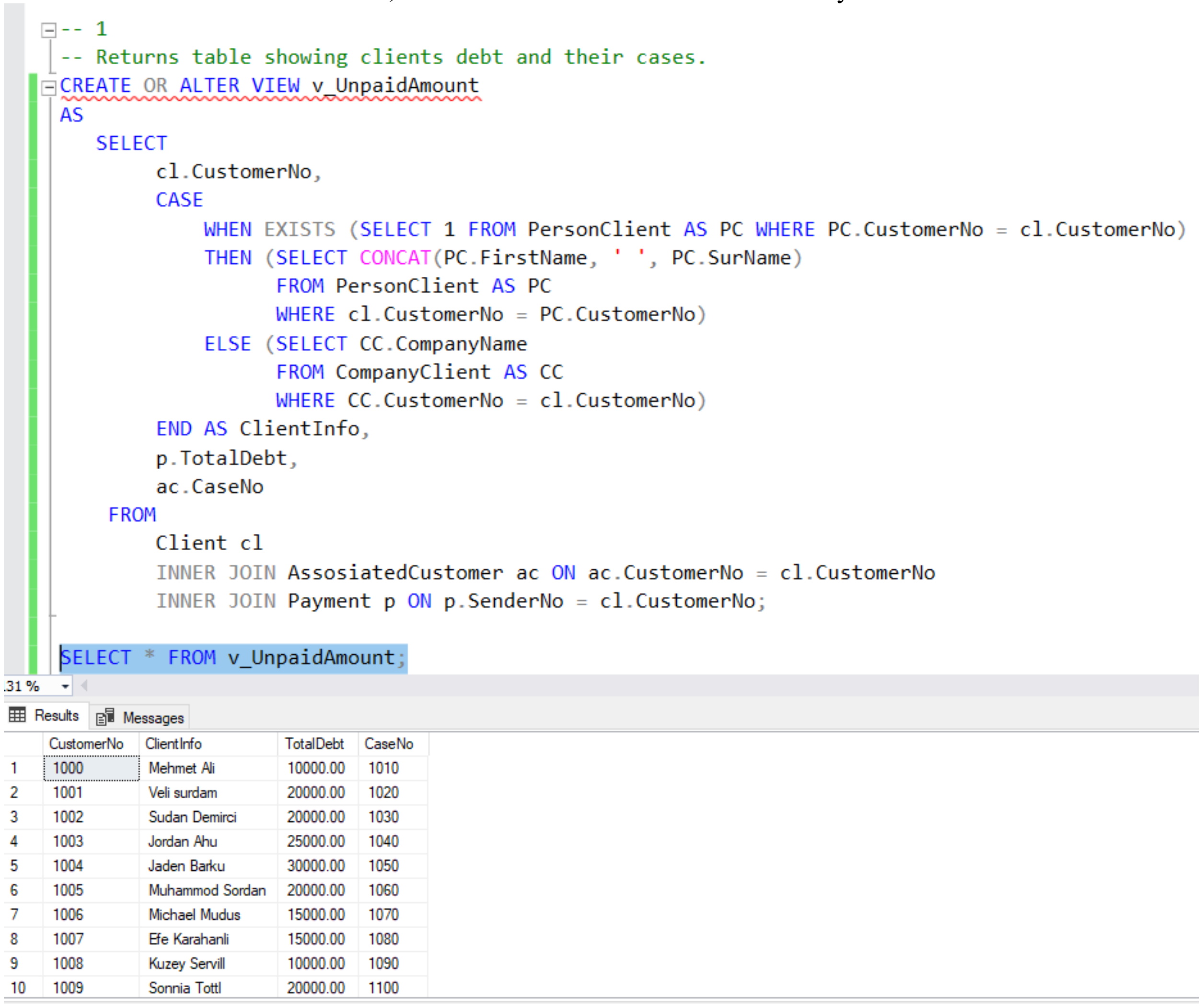


Results   Messages

	LawyerFullName	SuccessRate	ManagerFullName
1	Kosvu Kosova	1.00	Kara Demir
2	Fatih Balci	0.50	Ali Bali
3	Osman sanik	0.50	Ali Bali
4	Toru sevastapol	0.50	Ece Salu
5	Kilic Bulut	0.33	Kara Demir
6	Mehmet yakut	0.33	
7	Muhammed sajnsar	0.00	Ece Salu
8	Musa malibu	0.00	Ece Salu
9	Nura resit	0.00	Ali Bali

3- v_UnpaidAmount View

This view is used to see clients no, name and total debt and their cases they owe debt



```
-- 1
-- Returns table showing clients debt and their cases.
CREATE OR ALTER VIEW v_UnpaidAmount
AS
   SELECT
        cl.CustomerNo,
        CASE
            WHEN EXISTS (SELECT 1 FROM PersonClient AS PC WHERE PC.CustomerNo = cl.CustomerNo)
            THEN (SELECT CONCAT(PC.FirstName, ' ', PC.SurName)
                  FROM PersonClient AS PC
                  WHERE cl.CustomerNo = PC.CustomerNo)
            ELSE (SELECT CC.CompanyName
                  FROM CompanyClient AS CC
                  WHERE CC.CustomerNo = cl.CustomerNo)
        END AS ClientInfo,
        p.TotalDebt,
        ac.CaseNo
    FROM
        Client cl
        INNER JOIN AssosiatedCustomer ac ON ac.CustomerNo = cl.CustomerNo
        INNER JOIN Payment p ON p.SenderNo = cl.CustomerNo;

SELECT * FROM v_UnpaidAmount;
```

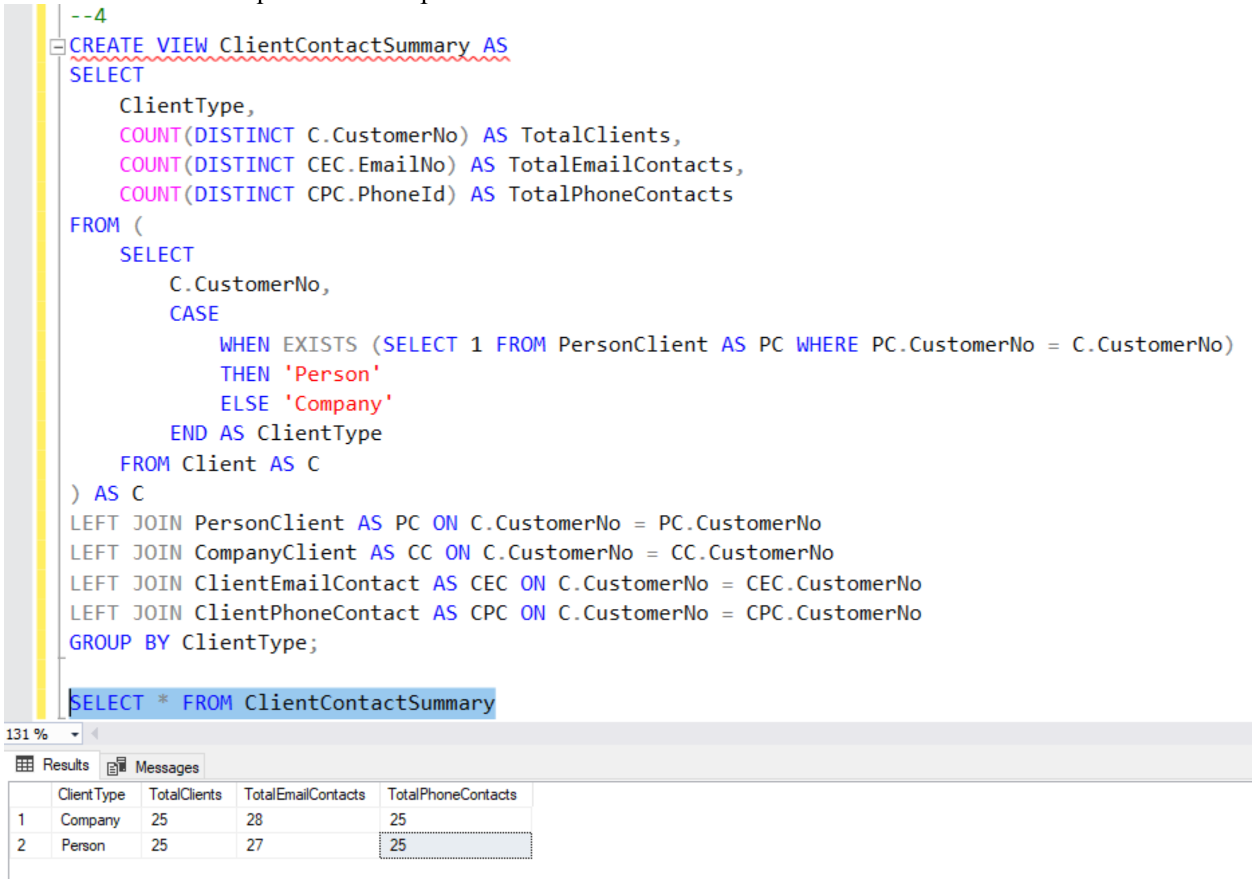
.31 %

Results | Messages

	CustomerNo	ClientInfo	TotalDebt	CaseNo
1	1000	Mehmet Ali	10000.00	1010
2	1001	Veli surdam	20000.00	1020
3	1002	Sudan Demirci	20000.00	1030
4	1003	Jordan Ahu	25000.00	1040
5	1004	Jaden Barku	30000.00	1050
6	1005	Muhammod Sordan	20000.00	1060
7	1006	Michael Mudus	15000.00	1070
8	1007	Efe Karahanli	15000.00	1080
9	1008	Kuzey Servill	10000.00	1090
10	1009	Sonnia Tottl	20000.00	1100

4- ClientContactSummary View

This view categorizes clients as either individuals or companies based on their presence in the PersonClient or CompanyClient tables. It then provides a summary of the total count of clients, along with the count of unique email and phone contacts associated with these clients.



```
--4
CREATE VIEW ClientContactSummary AS
SELECT
    ClientType,
    COUNT(DISTINCT C.CustomerNo) AS TotalClients,
    COUNT(DISTINCT CEC.EmailNo) AS TotalEmailContacts,
    COUNT(DISTINCT CPC.PhoneId) AS TotalPhoneContacts
FROM (
    SELECT
        C.CustomerNo,
        CASE
            WHEN EXISTS (SELECT 1 FROM PersonClient AS PC WHERE PC.CustomerNo = C.CustomerNo)
            THEN 'Person'
            ELSE 'Company'
        END AS ClientType
    FROM Client AS C
) AS C
LEFT JOIN PersonClient AS PC ON C.CustomerNo = PC.CustomerNo
LEFT JOIN CompanyClient AS CC ON C.CustomerNo = CC.CustomerNo
LEFT JOIN ClientEmailContact AS CEC ON C.CustomerNo = CEC.CustomerNo
LEFT JOIN ClientPhoneContact AS CPC ON C.CustomerNo = CPC.CustomerNo
GROUP BY ClientType;

SELECT * FROM ClientContactSummary
```

	ClientType	TotalClients	TotalEmailContacts	TotalPhoneContacts
1	Company	25	28	25
2	Person	25	27	25