**Assignment: Final Project**

IFT 530: Adv DB Management Systems

Author Names: Lokesh Lankalapalli, Suvethasree Poati Udhaya Kumar

Team Number: 36

ASU Student ID: 1231848932, 1232034065

Submission Instructions: Professor Ashish Gulati

Assignment Due Date: 12-06-2024

Script:

/\*

\*\* Author: Lokesh Lankalapalli, Suvethasree Poati Udhaya Kumar

\*\* Team Number : 36

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft Azure SQL Edge Developer (RTM) - 15.0.2000.1574 (ARM64)

\*\* Date Created: 12/06/2024

\*/

Code:

/\*

\*\* Author: Lokesh Lankalapalli, Suvethasree Poati Udhaya Kumar

\*\* Team Number : 36

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft Azure SQL Edge Developer (RTM) - 15.0.2000.1574 (ARM64)

\*\* Date Created: 12/06/2024

\*/

IF NOT EXISTS (SELECT \* FROM sys.databases WHERE name = 'TEAM\_36')

BEGIN

CREATE DATABASE TEAM\_36;

PRINT 'Database TEAM\_36 created successfully.';

END;

GO

-- Use the database after ensuring it exists

USE TEAM\_36;

GO

PRINT 'Using TEAM\_36 database.';

-- Check and create Tenants table

IF OBJECT\_ID('Tenants', 'U') IS NULL

BEGIN

CREATE TABLE Tenants (

TenantID INT NOT NULL PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) NOT NULL UNIQUE,

Phone VARCHAR(15)

);

PRINT 'Table Tenants created successfully.';

END;

-- Check and create Landlords table

IF OBJECT\_ID('Landlords', 'U') IS NULL

BEGIN

CREATE TABLE Landlords (

LandlordID INT NOT NULL PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) NOT NULL UNIQUE,

Phone VARCHAR(15)

);

PRINT 'Table Landlords created successfully.';

END;

-- Check and create Properties table

IF OBJECT\_ID('Properties', 'U') IS NULL

BEGIN

CREATE TABLE Properties (

PropertyID INT NOT NULL PRIMARY KEY,

LandlordID INT NOT NULL REFERENCES Landlords(LandlordID),

Address TEXT NOT NULL,

RentAmount DECIMAL(10, 2) NOT NULL,

Bedrooms INT NOT NULL,

Bathrooms INT NOT NULL,

Availability BIT NOT NULL

);

PRINT 'Table Properties created successfully.';

END;

-- Check and create FutureResidents table

IF OBJECT\_ID('FutureResidents', 'U') IS NULL

BEGIN

CREATE TABLE FutureResidents (

ResidentID INT NOT NULL PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) NOT NULL UNIQUE,

Phone VARCHAR(15),

PreferredMoveInDate DATE,

ApplicationStatus VARCHAR(50),

PropertyID INT NOT NULL REFERENCES Properties(PropertyID)

);

PRINT 'Table FutureResidents created successfully.';

END;

-- Check and create Leases table

IF OBJECT\_ID('Leases', 'U') IS NULL

BEGIN

CREATE TABLE Leases (

LeaseID INT NOT NULL PRIMARY KEY,

PropertyID INT NOT NULL REFERENCES Properties(PropertyID),

TenantID INT NOT NULL REFERENCES Tenants(TenantID),

StartDate DATE NOT NULL,

EndDate DATE,

MonthlyRent DECIMAL(10, 2) NOT NULL

);

PRINT 'Table Leases created successfully.';

END;

-- Check and create Payments table

IF OBJECT\_ID('Payments', 'U') IS NULL

BEGIN

CREATE TABLE Payments (

PaymentID INT NOT NULL PRIMARY KEY,

LeaseID INT NOT NULL REFERENCES Leases(LeaseID),

PaymentDate DATETIME NOT NULL,

Amount DECIMAL(10, 2) NOT NULL

);

PRINT 'Table Payments created successfully.';

END;

-- Check and create Employees table

IF OBJECT\_ID('Employees', 'U') IS NULL

BEGIN

CREATE TABLE Employees (

EmployeeID INT NOT NULL PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) NOT NULL UNIQUE,

Phone VARCHAR(15),

Position VARCHAR(50) NOT NULL,

HireDate DATE NOT NULL,

Salary DECIMAL(10, 2)

);

PRINT 'Table Employees created successfully.';

END;

-- Check and create TenantAudit table

IF OBJECT\_ID('TenantAudit', 'U') IS NULL

BEGIN

CREATE TABLE TenantAudit (

AuditID INT IDENTITY(1,1) PRIMARY KEY,

ActionPerformed VARCHAR(10),

TenantID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(15),

ActionDateTime DATETIME DEFAULT GETDATE()

);

PRINT 'Table TenantAudit created successfully.';

END;

-- Drop ActiveLeases view if it exists

IF OBJECT\_ID('ActiveLeases', 'V') IS NOT NULL

DROP VIEW ActiveLeases;

GO

--view 1

-- Drop ActiveLeases view if it exists

IF OBJECT\_ID('ActiveLeases', 'V') IS NOT NULL

DROP VIEW ActiveLeases;

GO

-- Create ActiveLeases view

CREATE VIEW ActiveLeases AS

SELECT

L.LeaseID,

L.StartDate,

L.EndDate,

L.MonthlyRent,

T.FirstName AS TenantFirstName,

T.LastName AS TenantLastName,

P.Address AS PropertyAddress,

P.RentAmount AS PropertyRent

FROM

Leases L

JOIN

Tenants T ON L.TenantID = T.TenantID

JOIN

Properties P ON L.PropertyID = P.PropertyID

WHERE

L.EndDate IS NULL OR L.EndDate > GETDATE(); -- Active leases only

GO

--view 2

-- Drop TenantPaymentHistory view if it exists

IF OBJECT\_ID('TenantPaymentHistory', 'V') IS NOT NULL

DROP VIEW TenantPaymentHistory;

GO

-- Create TenantPaymentHistory view

CREATE VIEW TenantPaymentHistory AS

SELECT

T.TenantID,

T.FirstName,

T.LastName,

L.LeaseID,

L.StartDate,

L.EndDate,

P.PaymentDate,

P.Amount AS PaymentAmount

FROM

Payments P

JOIN

Leases L ON P.LeaseID = L.LeaseID

JOIN

Tenants T ON L.TenantID = T.TenantID

WHERE

P.PaymentDate >= '2024-01-01'; -- Payments from January 1st, 2024 onwards

GO

--view 3

-- Drop AvailableProperties view if it exists

IF OBJECT\_ID('AvailableProperties', 'V') IS NOT NULL

DROP VIEW AvailableProperties;

GO

-- Create AvailableProperties view

CREATE VIEW AvailableProperties AS

SELECT

P.PropertyID,

P.Address,

P.RentAmount,

P.Bedrooms,

P.Availability,

L.FirstName AS LandlordFirstName,

L.LastName AS LandlordLastName,

L.Email AS LandlordEmail

FROM

Properties P

JOIN

Landlords L ON P.LandlordID = L.LandlordID

WHERE

P.Availability = 1; -- Only available properties

GO

-- Print ActiveLeases view

PRINT 'ActiveLeases View:';

SELECT \* FROM ActiveLeases;

-- Print TenantPaymentHistory view

PRINT 'TenantPaymentHistory View:';

SELECT \* FROM TenantPaymentHistory;

-- Print AvailableProperties view

PRINT 'AvailableProperties View:';

SELECT \* FROM AvailableProperties;

-- Create TenantAudit table

IF OBJECT\_ID('TenantAudit', 'U') IS NULL

BEGIN

CREATE TABLE TenantAudit (

AuditID INT IDENTITY(1,1) PRIMARY KEY,

ActionPerformed VARCHAR(10),

TenantID INT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(15),

ActionDateTime DATETIME DEFAULT GETDATE()

);

PRINT 'Table TenantAudit created successfully.';

END;

-- Create trigger for INSERT on Tenants

IF OBJECT\_ID('trg\_Tenants\_Insert', 'TR') IS NOT NULL

DROP TRIGGER trg\_Tenants\_Insert;

GO

CREATE TRIGGER trg\_Tenants\_Insert

ON Tenants

AFTER INSERT

AS

BEGIN

INSERT INTO TenantAudit (ActionPerformed, TenantID, FirstName, LastName, Email, Phone)

SELECT 'INSERT', TenantID, FirstName, LastName, Email, Phone

FROM inserted;

END;

GO

-- Create trigger for UPDATE on Tenants

IF OBJECT\_ID('trg\_Tenants\_Update', 'TR') IS NOT NULL

DROP TRIGGER trg\_Tenants\_Update;

GO

CREATE TRIGGER trg\_Tenants\_Update

ON Tenants

AFTER UPDATE

AS

BEGIN

INSERT INTO TenantAudit (ActionPerformed, TenantID, FirstName, LastName, Email, Phone)

SELECT 'UPDATE', TenantID, FirstName, LastName, Email, Phone

FROM inserted;

END;

GO

-- Create trigger for DELETE on Tenants

IF OBJECT\_ID('trg\_Tenants\_Delete', 'TR') IS NOT NULL

DROP TRIGGER trg\_Tenants\_Delete;

GO

CREATE TRIGGER trg\_Tenants\_Delete

ON Tenants

AFTER DELETE

AS

BEGIN

INSERT INTO TenantAudit (ActionPerformed, TenantID, FirstName, LastName, Email, Phone)

SELECT 'DELETE', TenantID, FirstName, LastName, Email, Phone

FROM deleted;

END;

GO

-- Insert operation

INSERT INTO Tenants (TenantID, FirstName, LastName, Email, Phone)

VALUES (11, 'Sarah', 'Connor', 'sarahconnor@example.com', '1234567890');

GO

-- Update operation

UPDATE Tenants

SET Phone = '0987654321'

WHERE TenantID = 11;

GO

-- Delete operation

DELETE FROM Tenants

WHERE TenantID = 11;

GO

-- View the audit log

SELECT \* FROM TenantAudit;

-- Drop Stored Procedure if it exists

IF OBJECT\_ID('GetTenantDetails', 'P') IS NOT NULL

DROP PROCEDURE GetTenantDetails;

GO

-- Create Stored Procedure: GetTenantDetails

CREATE PROCEDURE GetTenantDetails

@TenantID INT

AS

BEGIN

SELECT

T.TenantID,

T.FirstName,

T.LastName,

T.Email,

T.Phone,

L.LeaseID,

L.StartDate,

L.EndDate,

L.MonthlyRent,

P.PaymentDate,

P.Amount AS PaymentAmount

FROM

Tenants T

LEFT JOIN

Leases L ON T.TenantID = L.TenantID

LEFT JOIN

Payments P ON L.LeaseID = P.LeaseID

WHERE

T.TenantID = @TenantID;

END;

GO

-- Test Stored Procedure

EXEC GetTenantDetails @TenantID = 1;

-- Drop UDF if it exists

IF OBJECT\_ID('CalculateTotalPayments', 'FN') IS NOT NULL

DROP FUNCTION CalculateTotalPayments;

GO

-- Create UDF: CalculateTotalPayments

CREATE FUNCTION CalculateTotalPayments(@TenantID INT)

RETURNS DECIMAL(10, 2)

AS

BEGIN

DECLARE @TotalPayments DECIMAL(10, 2);

SELECT

@TotalPayments = SUM(P.Amount)

FROM

Payments P

JOIN

Leases L ON P.LeaseID = L.LeaseID

WHERE

L.TenantID = @TenantID;

RETURN ISNULL(@TotalPayments, 0);

END;

GO

-- Test UDF

SELECT dbo.CalculateTotalPayments(1) AS TotalPayments;

-- Declare a cursor for tenants

DECLARE TenantCursor CURSOR FOR

SELECT TenantID, FirstName, LastName

FROM Tenants;

-- Declare variables to store tenant details

DECLARE @TenantID INT, @FirstName VARCHAR(50), @LastName VARCHAR(50);

-- Open the cursor

OPEN TenantCursor;

-- Fetch the first row from the cursor

FETCH NEXT FROM TenantCursor INTO @TenantID, @FirstName, @LastName;

-- Loop through the cursor and print tenant details

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Tenant ID: ' + CAST(@TenantID AS VARCHAR) + ', Name: ' + @FirstName + ' ' + @LastName;

-- Fetch the next row

FETCH NEXT FROM TenantCursor INTO @TenantID, @FirstName, @LastName;

END;

-- Close and deallocate the cursor

CLOSE TenantCursor;

DEALLOCATE TenantCursor;

-- Delete all rows from all tables

DELETE FROM Payments;

DELETE FROM Leases;

DELETE FROM FutureResidents;

DELETE FROM Properties;

DELETE FROM Landlords;

DELETE FROM Tenants;

DELETE FROM Employees;

-- Populate Tenants table (10 rows)

INSERT INTO Tenants (TenantID, FirstName, LastName, Email, Phone)

VALUES

(1, 'John', 'Doe', 'johndoe@example.com', '1234567890'),

(2, 'Jane', 'Smith', 'janesmith@example.com', '0987654321'),

(3, 'Mike', 'Brown', 'mikebrown@example.com', '1122334455'),

(4, 'Emily', 'Davis', 'emilydavis@example.com', '6677889900'),

(5, 'Chris', 'Johnson', 'chrisjohnson@example.com', '5566778899'),

(6, 'Anna', 'White', 'annawhite@example.com', '4433221100'),

(7, 'Tom', 'Harris', 'tomharris@example.com', '7788990011'),

(8, 'Laura', 'Taylor', 'laurataylor@example.com', '2233445566'),

(9, 'James', 'Anderson', 'jamesanderson@example.com', '8899001122'),

(10, 'Olivia', 'Thomas', 'oliviathomas@example.com', '3344556677');

-- Populate Landlords table (10 rows)

INSERT INTO Landlords (LandlordID, FirstName, LastName, Email, Phone)

VALUES

(1, 'Robert', 'Miller', 'robertmiller@example.com', '1231231234'),

(2, 'Sophia', 'Wilson', 'sophiawilson@example.com', '2342342345'),

(3, 'William', 'Moore', 'williammoore@example.com', '3453453456'),

(4, 'Mia', 'Taylor', 'miataylor@example.com', '4564564567'),

(5, 'Lucas', 'Clark', 'lucasclark@example.com', '5675675678'),

(6, 'Ethan', 'Lewis', 'ethanlewis@example.com', '6786786789'),

(7, 'Emma', 'Hall', 'emmahall@example.com', '7897897890'),

(8, 'Alexander', 'King', 'alexanderking@example.com', '8908908901'),

(9, 'Isabella', 'Scott', 'isabellascott@example.com', '9019019012'),

(10, 'Charlotte', 'Green', 'charlottegreen@example.com', '0120120123');

-- Populate Properties table (10 rows)

INSERT INTO Properties (PropertyID, LandlordID, Address, RentAmount, Bedrooms, Bathrooms, Availability)

VALUES

(1, 1, '123 Main St, Apt 101', 1200.00, 2, 1, 1),

(2, 2, '123 Main St, Apt 102', 1500.00, 3, 2, 1),

(3, 3, '123 Main St, Apt 103', 1000.00, 1, 1, 1),

(4, 4, '123 Main St, Apt 104', 1800.00, 4, 3, 0),

(5, 5, '123 Main St, Apt 105', 1300.00, 2, 2, 1),

(6, 6, '123 Main St, Apt 106', 1100.00, 1, 1, 0),

(7, 7, '123 Main St, Apt 107', 1400.00, 2, 2, 1),

(8, 8, '123 Main St, Apt 108', 1700.00, 3, 3, 1),

(9, 9, '123 Main St, Apt 109', 900.00, 1, 1, 1),

(10, 10, '123 Main St, Apt 110', 2000.00, 4, 3, 0);

-- Populate FutureResidents table (10 rows)

INSERT INTO FutureResidents (ResidentID, FirstName, LastName, Email, Phone, PreferredMoveInDate, ApplicationStatus, PropertyID)

VALUES

(1, 'Sarah', 'Connor', 'sarahconnor@example.com', '1234567890', '2024-06-01', 'Pending', 1),

(2, 'Kyle', 'Reese', 'kylereese@example.com', '9876543210', '2024-07-01', 'Approved', 2),

(3, 'Ellen', 'Ripley', 'ellenripley@example.com', '2345678901', '2024-05-15', 'Pending', 3),

(4, 'John', 'Wick', 'johnwick@example.com', '8765432109', '2024-08-01', 'Approved', 4),

(5, 'Diana', 'Prince', 'dianaprince@example.com', '3456789012', '2024-09-01', 'Pending', 5),

(6, 'Clark', 'Kent', 'clarkkent@example.com', '7654321098', '2024-04-01', 'Approved', 6),

(7, 'Bruce', 'Wayne', 'brucewayne@example.com', '4567890123', '2024-03-01', 'Pending', 7),

(8, 'Tony', 'Stark', 'tonystark@example.com', '6543210987', '2024-10-01', 'Approved', 8),

(9, 'Natasha', 'Romanoff', 'natasharomanoff@example.com', '5678901234', '2024-11-01', 'Pending', 9),

(10, 'Steve', 'Rogers', 'steverogers@example.com', '5432109876', '2024-12-01', 'Approved', 10);

-- Populate Employees table (10 rows)

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Phone, Position, HireDate, Salary)

VALUES

(1, 'Alice', 'Johnson', 'alicejohnson@example.com', '1234567890', 'Leasing Manager', '2020-05-01', 60000.00),

(2, 'Bob', 'Smith', 'bobsmith@example.com', '9876543210', 'Property Manager', '2021-03-15', 55000.00),

(3, 'Charlie', 'Brown', 'charliebrown@example.com', '2345678901', 'Maintenance Staff', '2022-06-20', 40000.00),

(4, 'David', 'Taylor', 'davidtaylor@example.com', '8765432109', 'Accountant', '2019-11-01', 50000.00),

(5, 'Emma', 'Wilson', 'emmawilson@example.com', '3456789012', 'Receptionist', '2023-01-05', 35000.00),

(6, 'Frank', 'Martin', 'frankmartin@example.com', '7654321098', 'Security Guard', '2022-02-28', 32000.00),

(7, 'Grace', 'Lee', 'gracelee@example.com', '4567890123', 'Leasing Agent', '2020-08-10', 45000.00),

(8, 'Hannah', 'Moore', 'hannahmoore@example.com', '6543210987', 'Marketing Specialist', '2021-12-15', 48000.00),

(9, 'Ian', 'Wright', 'ianwright@example.com', '5678901234', 'Customer Support', '2022-04-25', 37000.00),

(10, 'Julia', 'Evans', 'juliaevans@example.com', '5432109876', 'IT Specialist', '2020-07-01', 65000.00);

-- Populate Leases table (20 rows)

INSERT INTO Leases (LeaseID, PropertyID, TenantID, StartDate, EndDate, MonthlyRent)

VALUES

(1, 1, 1, '2024-01-01', '2024-12-31', 1200.00),

(2, 2, 2, '2024-02-01', '2025-01-31', 1500.00),

(3, 3, 3, '2024-03-01', NULL, 1000.00),

(4, 4, 4, '2024-04-01', '2024-10-31', 1800.00),

(5, 5, 5, '2024-05-01', NULL, 1300.00),

(6, 6, 6, '2024-06-01', '2024-11-30', 1100.00),

(7, 7, 7, '2024-07-01', NULL, 1400.00),

(8, 8, 8, '2024-08-01', '2025-07-31', 1700.00),

(9, 9, 9, '2024-09-01', NULL, 900.00),

(10, 10, 10, '2024-10-01', NULL, 2000.00),

(11, 1, 1, '2025-01-01', '2025-12-31', 1200.00),

(12, 2, 2, '2025-02-01', '2026-01-31', 1500.00),

(13, 3, 3, '2025-03-01', NULL, 1000.00),

(14, 4, 4, '2025-04-01', '2025-10-31', 1800.00),

(15, 5, 5, '2025-05-01', NULL, 1300.00),

(16, 6, 6, '2025-06-01', '2025-11-30', 1100.00),

(17, 7, 7, '2025-07-01', NULL, 1400.00),

(18, 8, 8, '2025-08-01', '2026-07-31', 1700.00),

(19, 9, 9, '2025-09-01', NULL, 900.00),

(20, 10, 10, '2025-10-01', NULL, 2000.00);

-- Populate Payments table (30 rows)

INSERT INTO Payments (PaymentID, LeaseID, PaymentDate, Amount)

VALUES

(1, 1, '2024-01-05', 1200.00),

(2, 2, '2024-02-05', 1500.00),

(3, 3, '2024-03-05', 1000.00),

(4, 4, '2024-04-05', 1800.00),

(5, 5, '2024-05-05', 1300.00),

(6, 6, '2024-06-05', 1100.00),

(7, 7, '2024-07-05', 1400.00),

(8, 8, '2024-08-05', 1700.00),

(9, 9, '2024-09-05', 900.00),

(10, 10, '2024-10-05', 2000.00),

(11, 1, '2024-02-05', 1200.00),

(12, 2, '2024-03-05', 1500.00),

(13, 3, '2024-04-05', 1000.00),

(14, 4, '2024-05-05', 1800.00),

(15, 5, '2024-06-05', 1300.00),

(16, 6, '2024-07-05', 1100.00),

(17, 7, '2024-08-05', 1400.00),

(18, 8, '2024-09-05', 1700.00),

(19, 9, '2024-10-05', 900.00),

(20, 10, '2024-11-05', 2000.00),

(21, 1, '2025-01-05', 1200.00),

(22, 2, '2025-02-05', 1500.00),

(23, 3, '2025-03-05', 1000.00),

(24, 4, '2025-04-05', 1800.00),

(25, 5, '2025-05-05', 1300.00),

(26, 6, '2025-06-05', 1100.00),

(27, 7, '2025-07-05', 1400.00),

(28, 8, '2025-08-05', 1700.00),

(29, 9, '2025-09-05', 900.00),

(30, 10, '2025-10-05', 2000.00);

-- Print Tenants table

PRINT 'Tenants Table:';

SELECT \* FROM Tenants;

-- Print Landlords table

PRINT 'Landlords Table:';

SELECT \* FROM Landlords;

-- Print Properties table

PRINT 'Properties Table:';

SELECT \* FROM Properties;

-- Print FutureResidents table

PRINT 'FutureResidents Table:';

SELECT \* FROM FutureResidents;

-- Print Leases table

PRINT 'Leases Table:';

SELECT \* FROM Leases;

-- Print Payments table

PRINT 'Payments Table:';

SELECT \* FROM Payments;

-- Print Employees table

PRINT 'Employees Table:';

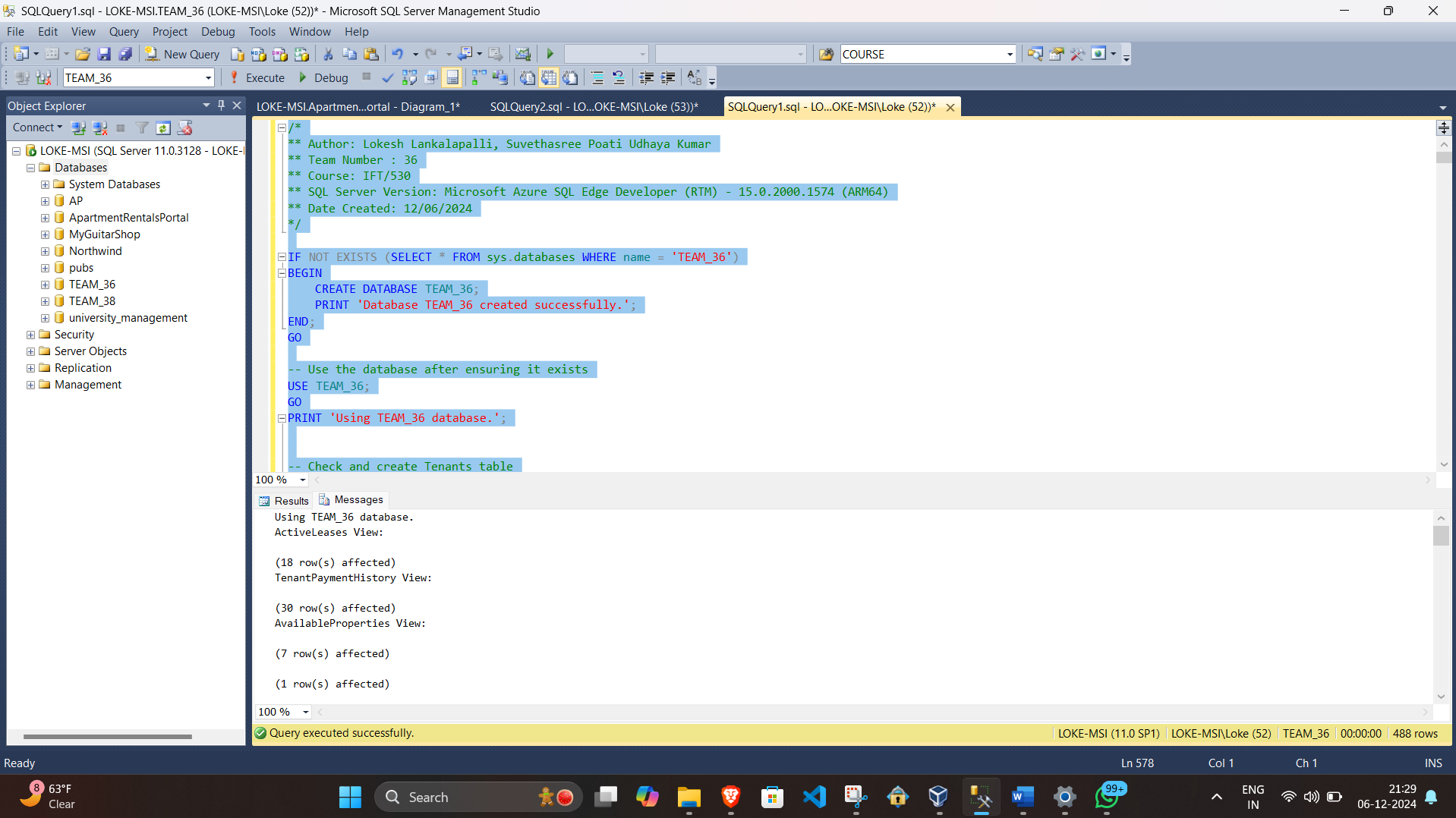
SELECT \* FROM Employees;

-- Print TenantAudit table

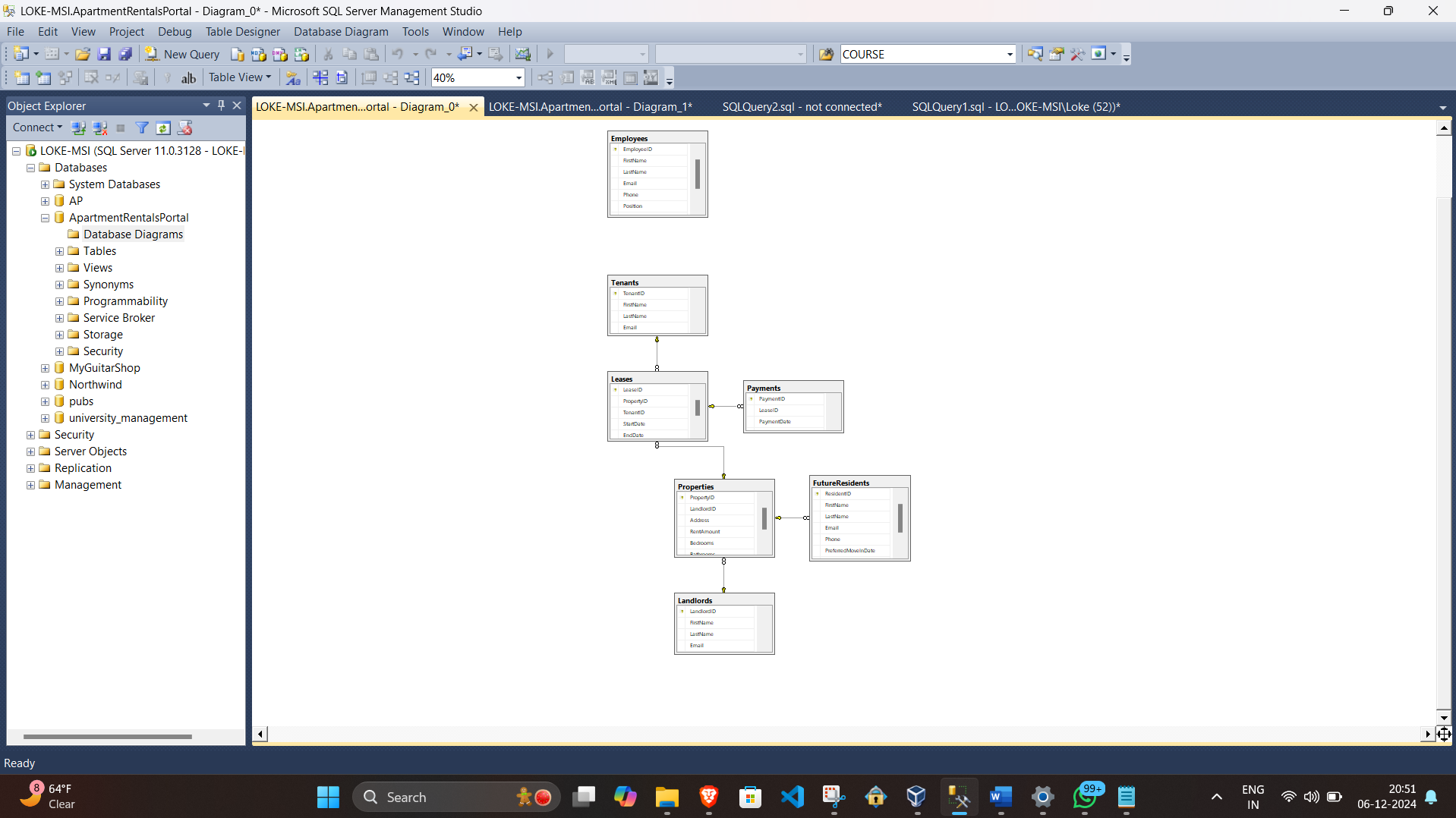
PRINT 'TenantAudit Table:';

SELECT \* FROM TenantAudit;

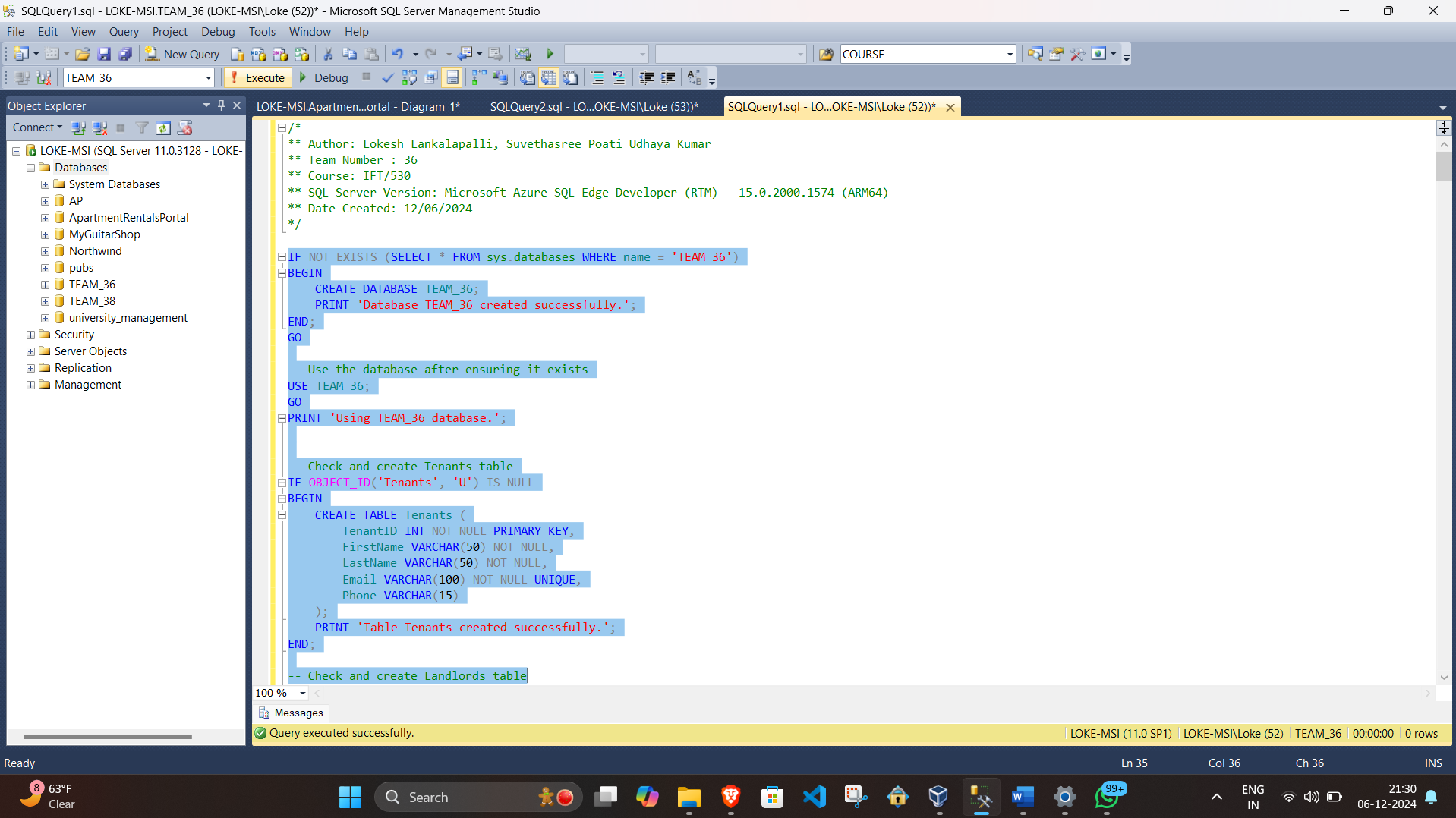
Code Execution:



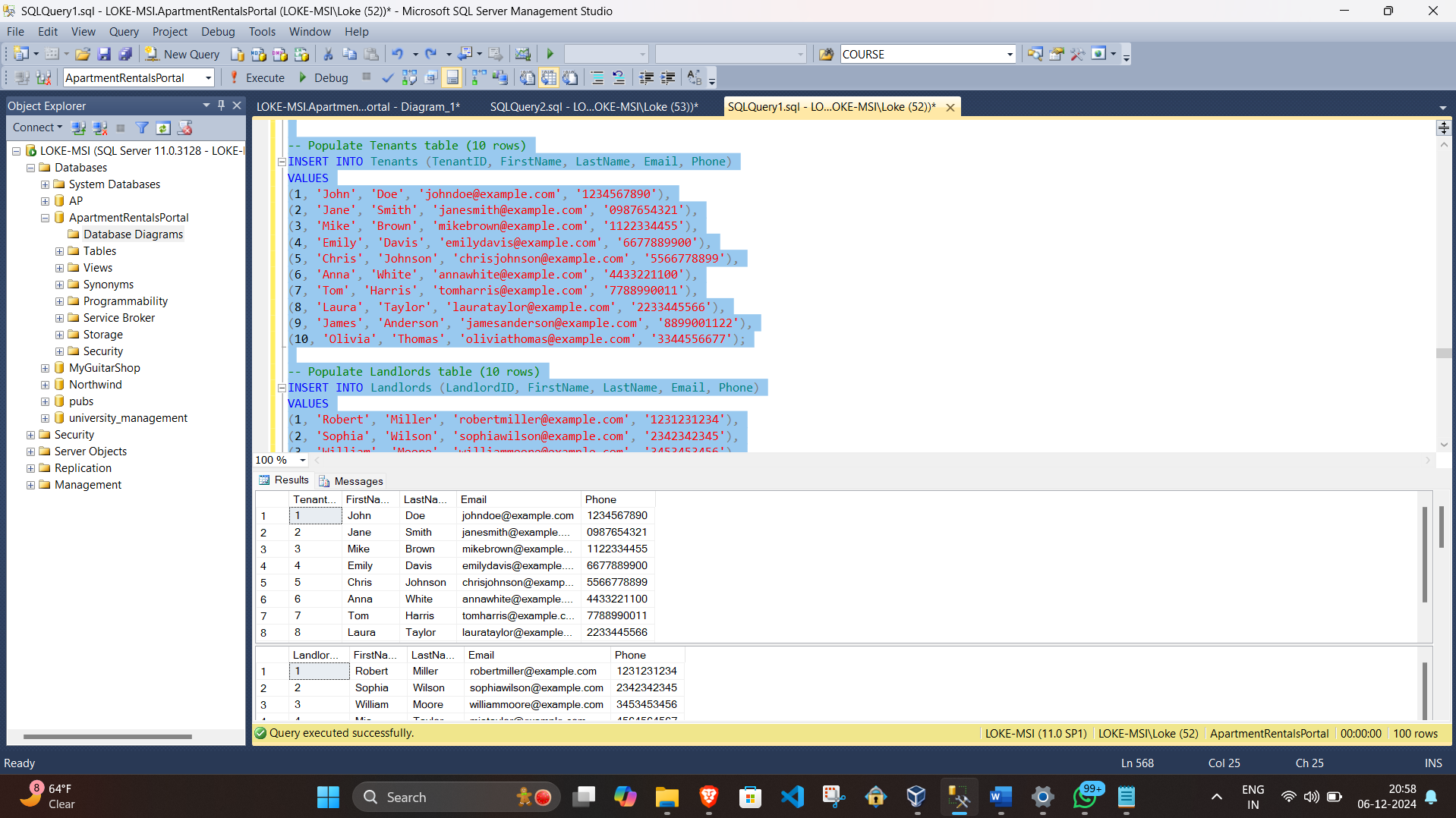
Entity Relation Diagram:



1. Creating Tables:



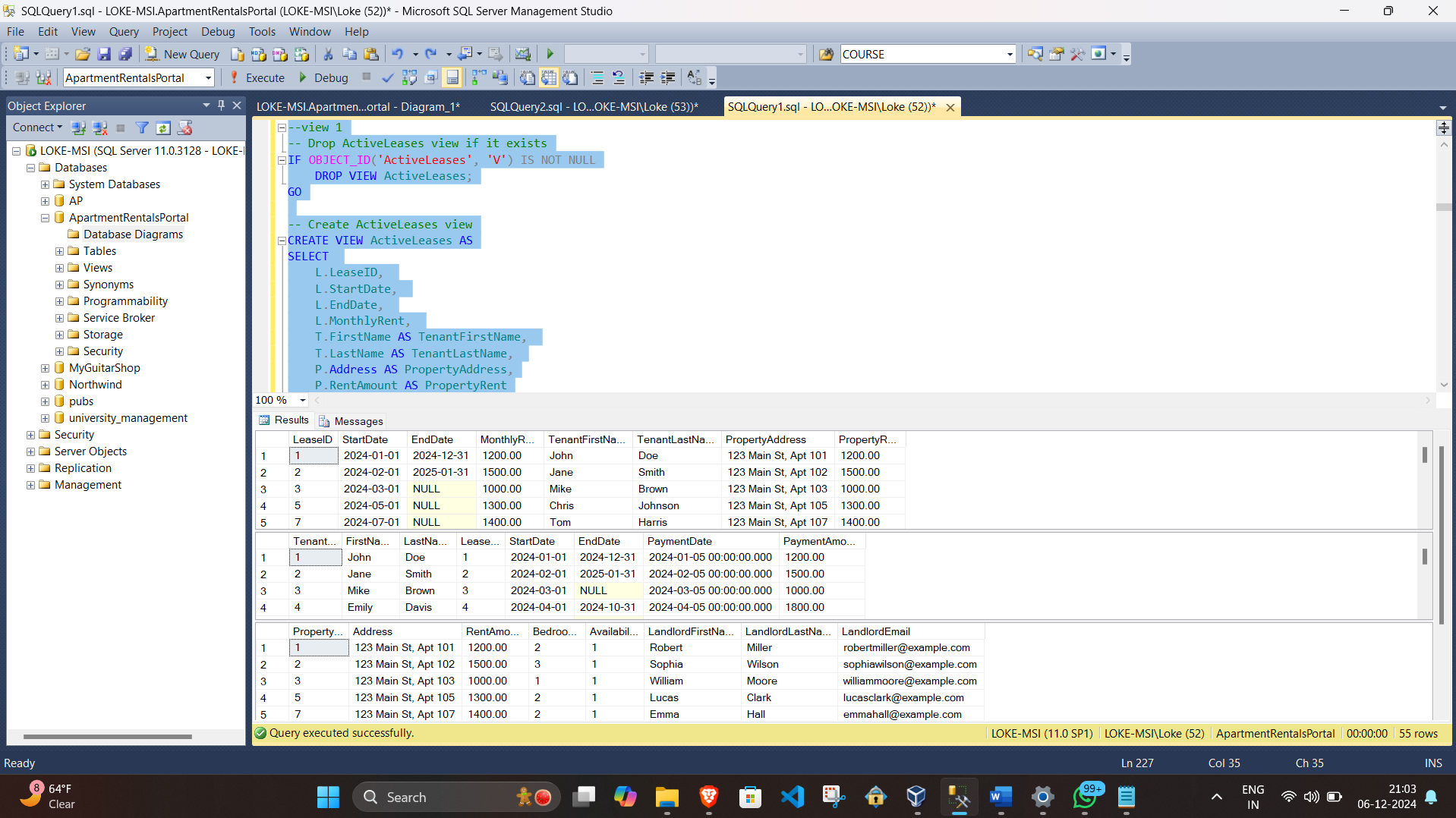
1. Populating Tables:



The script populates each dimension table (Tenants, Landlords, Properties, Employees) with 10 rows each, adhering to the requirement. For transactional tables like Leases and Payments, the script includes 20 rows for Leases and 30 rows for Payments, ensuring transactional data is populated adequately.

This setup ensures that the database is well-seeded with representative data, allowing for meaningful queries, views, and testing of business logic.

1. **Creating Views:**



Views Created:

ActiveLeases:

Query Objective: Displays active lease details, including tenant and property information. This view is useful for property managers to monitor ongoing leases.

Key Features: Joins Leases, Tenants, and Properties tables and filters for leases with no end date or end dates in the future.

TenantPaymentHistory:

Query Objective: Lists payment history for tenants, including details of lease and payment dates. It is useful for tracking tenant payments and outstanding dues.

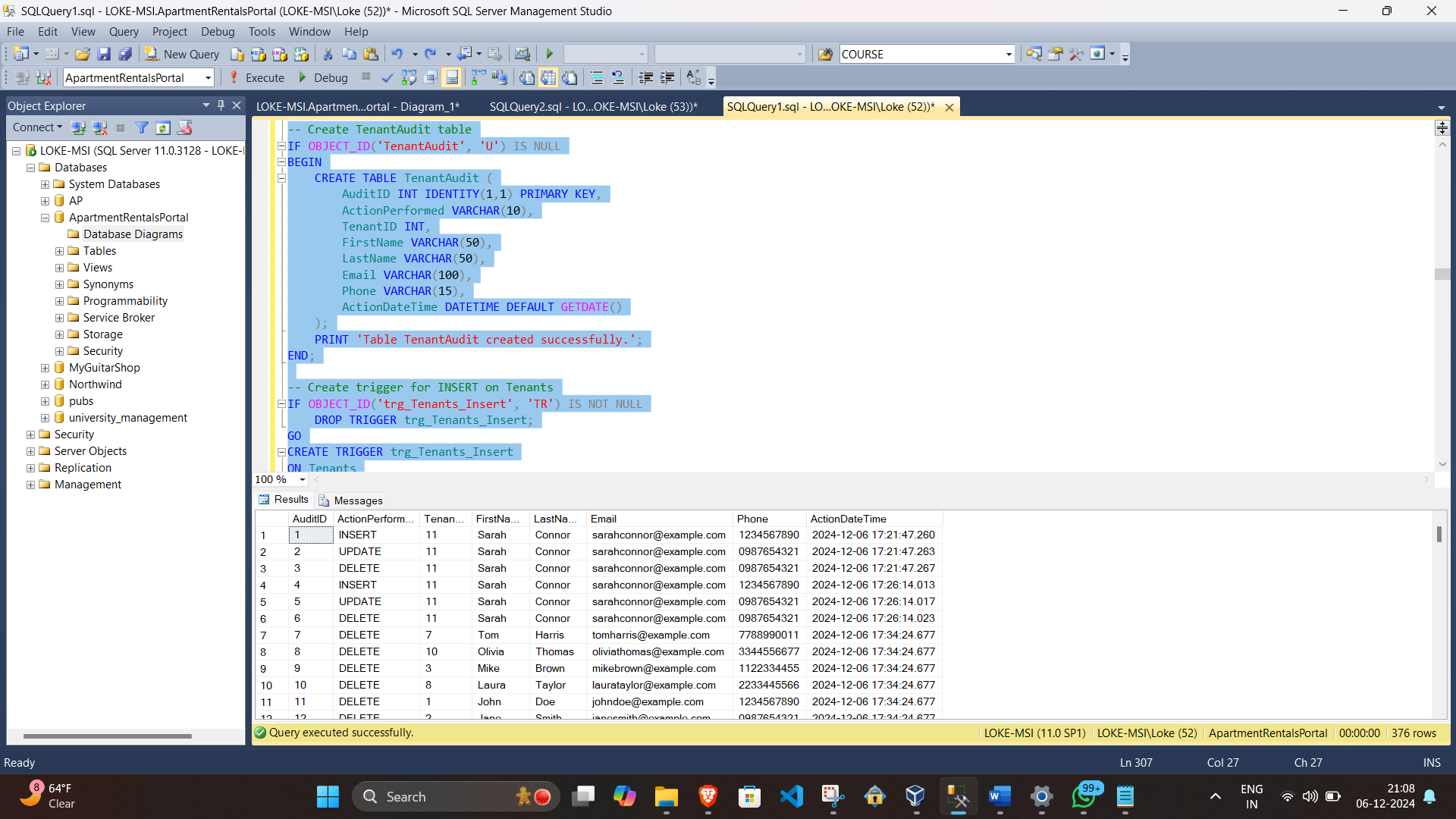
Key Features: Joins Payments, Leases, and Tenants and restricts data to payments made after January 1, 2024.

AvailableProperties:

Query Objective: Provides a list of currently available properties along with landlord contact details. This view is useful for marketing or allocating available properties to new tenants.

Key Features: Joins Properties and Landlords and filters for properties marked as available (Availability = 1).

1. **Create Audit Table:**

****

Description:

An audit table (TenantAudit) is created to track changes to the Tenants table. The audit table includes:

Columns: Action performed (INSERT, UPDATE, or DELETE), tenant details, and a timestamp (ActionDateTime) for when the change occurred.

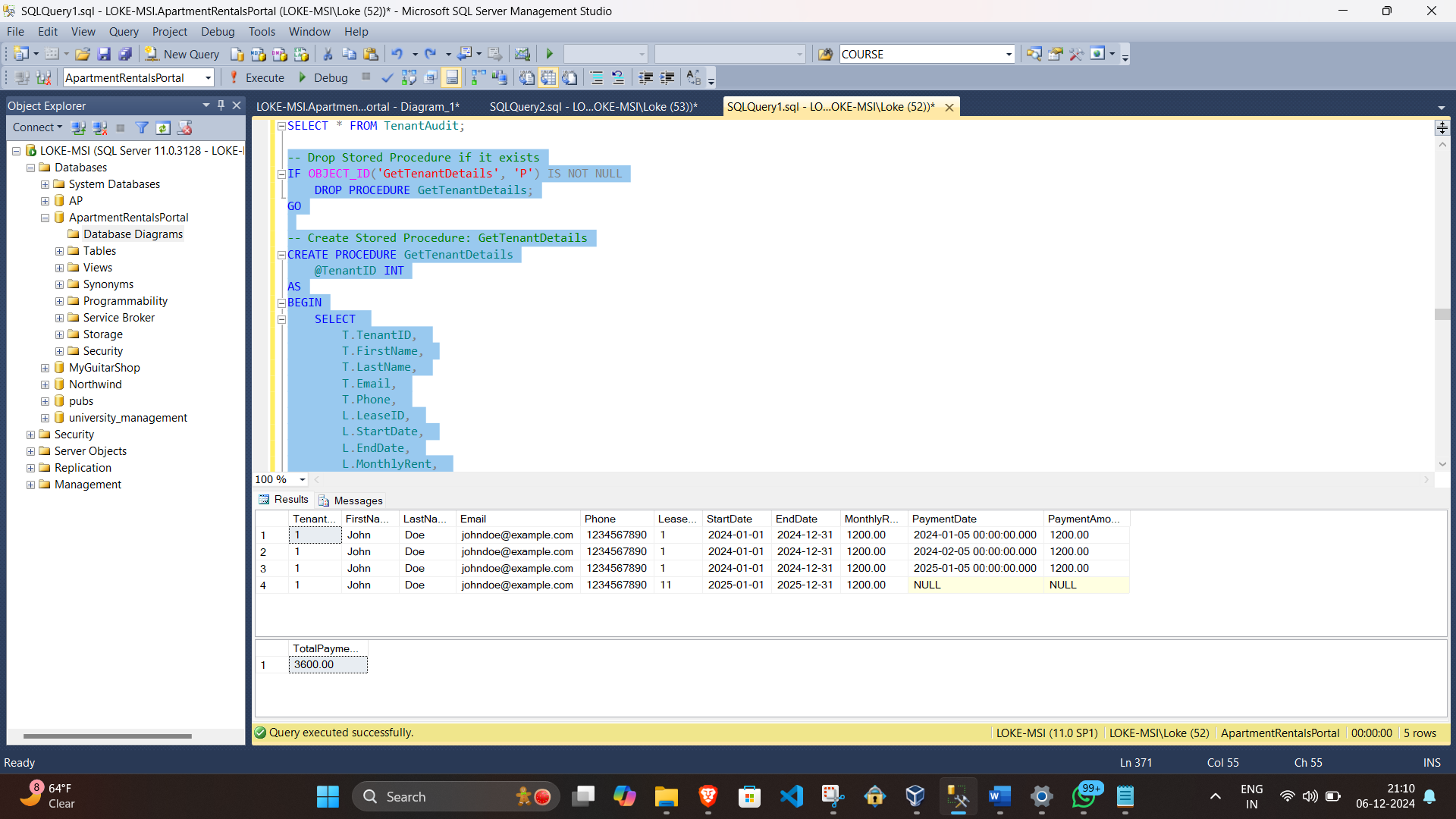
Triggers:

trg\_Tenants\_Insert, trg\_Tenants\_Update, and trg\_Tenants\_Delete track insertions, updates, and deletions, respectively, in the Tenants table.

Demonstration:

Test cases are provided to insert a new tenant, update their phone number, and delete the tenant. The TenantAudit table captures each action, verifying the functionality of the triggers.

1. **Stored Procedures and User Defined Functions:**



Stored Procedure: GetTenantDetails:

Objective: Retrieves a tenant’s details along with their lease and payment history based on the provided tenant ID.

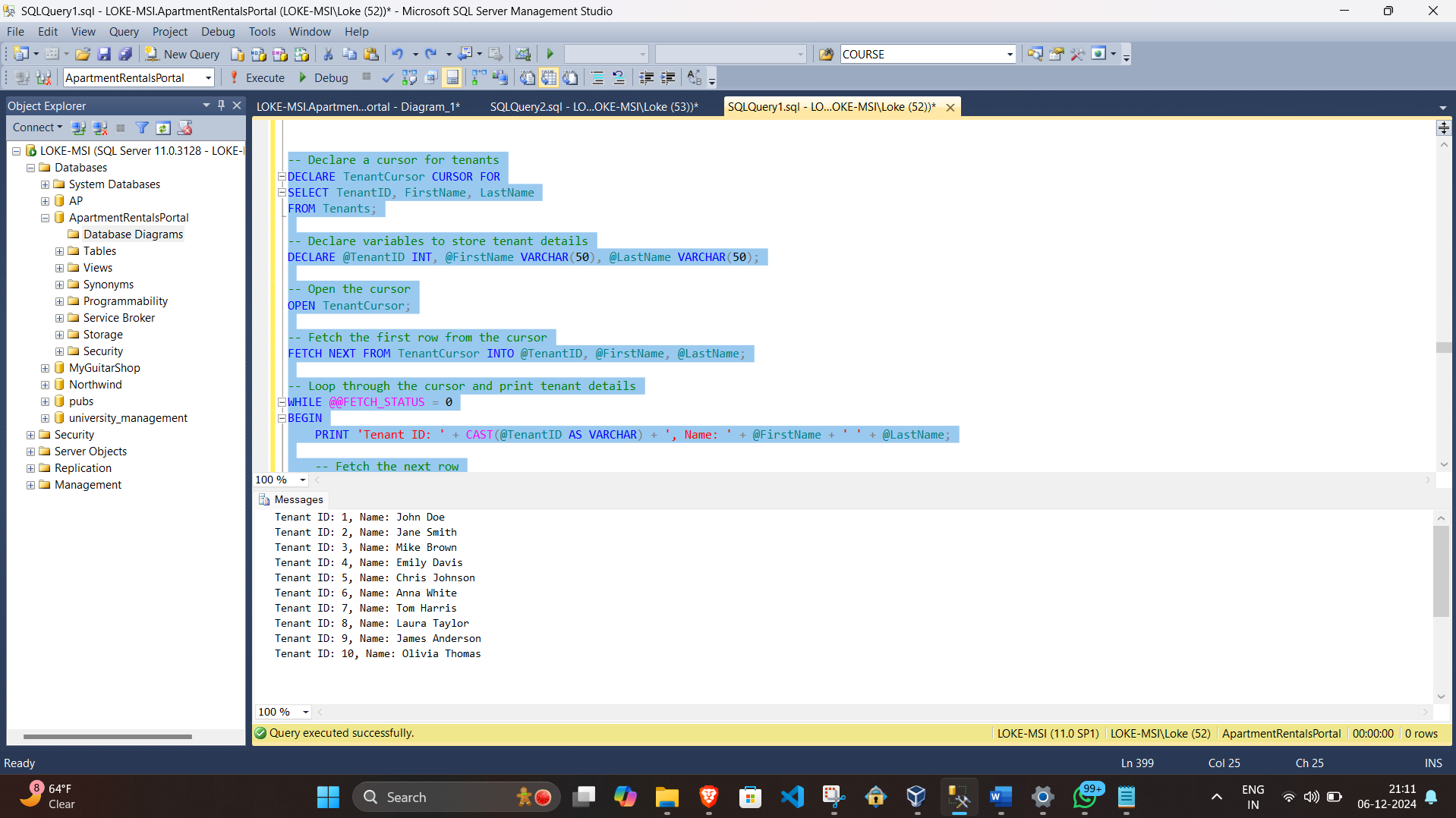
Demonstration: A test case is included to call the stored procedure for a specific tenant (TenantID = 1) and retrieve relevant data. Create and drop scripts are also provided.

User Defined Function: CalculateTotalPayments:

Objective: Calculates the total payments made by a tenant.

Demonstration: A test case is included to calculate the total payments for TenantID = 1. The function uses SUM over the Payments table for the specific tenant.

1. **Cursor:**



Cursor: TenantCursor:

Objective: Iterates through the Tenants table and prints the tenant IDs and names.

Demonstration:

Declares a cursor (TenantCursor) to fetch tenant details.

Opens the cursor, fetches tenant data row by row, and prints their details.

Ensures proper closure and deallocation of the cursor after use.