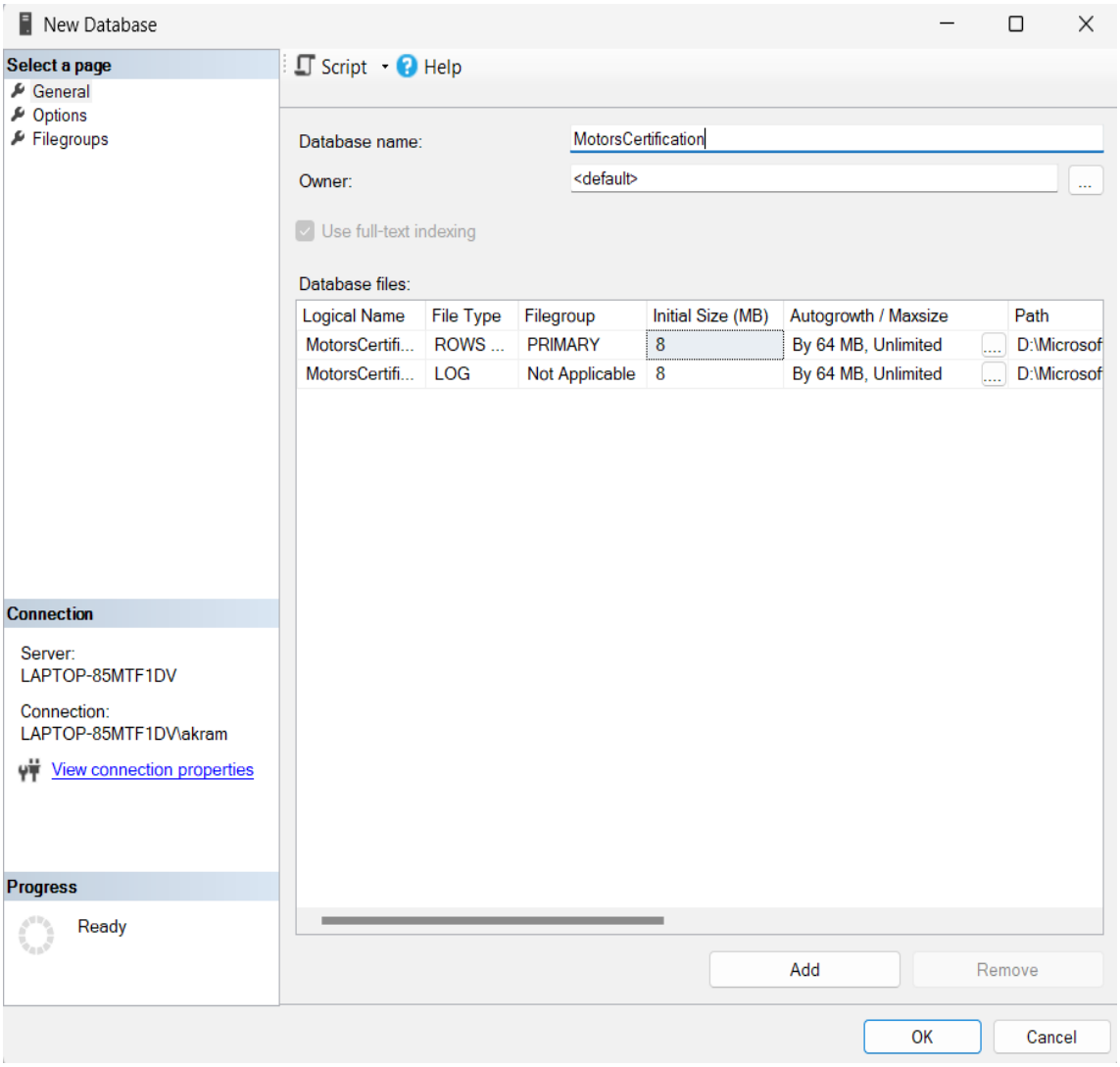
Certification Project II

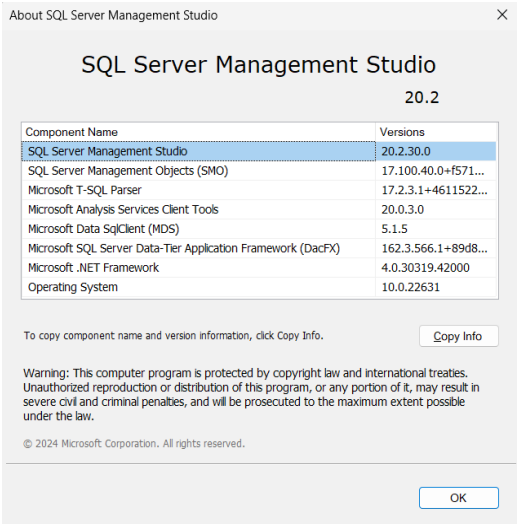
Microsoft SQL Server

Internship Program: Data Science and Machine Learning Program
Student: Akram M'Tir
Date: March 2025

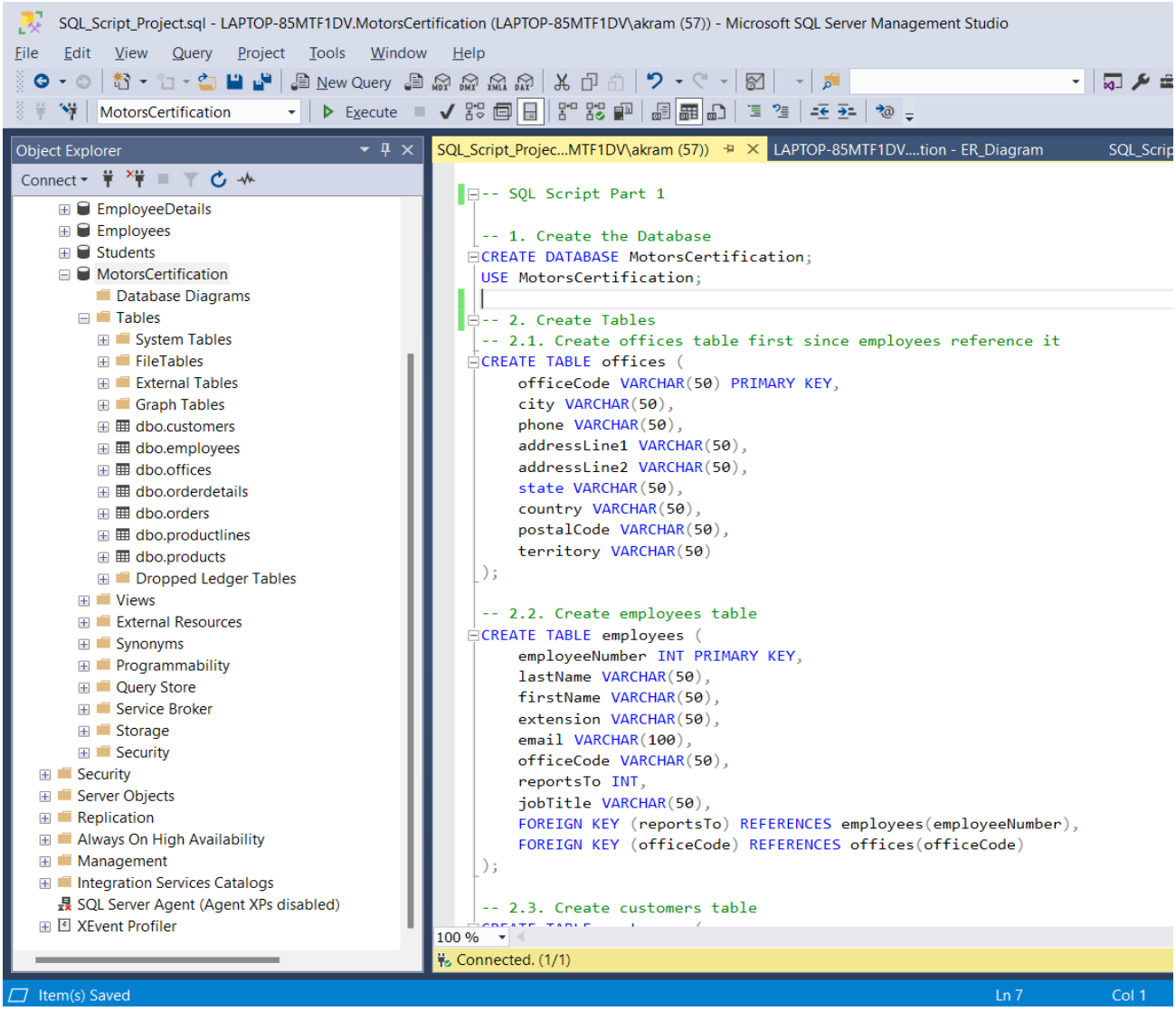
Table of Contents

Database creation

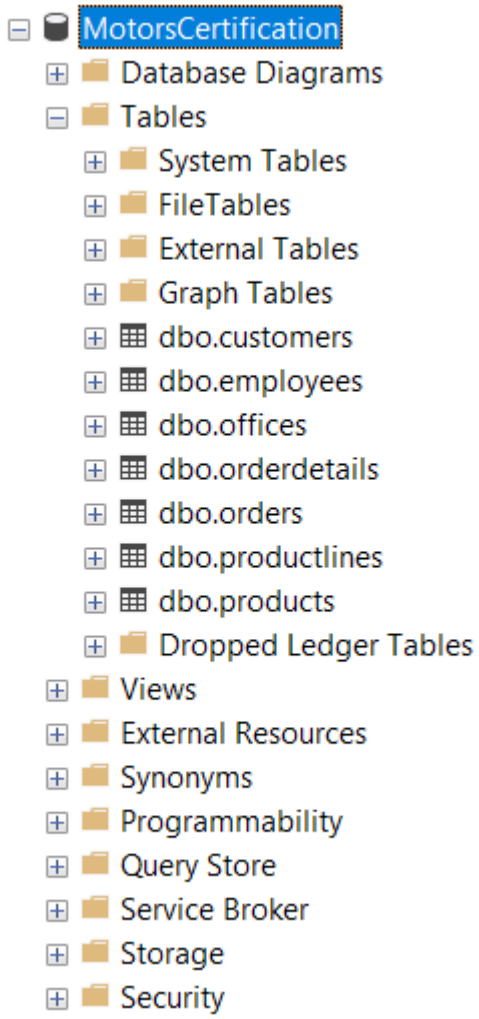




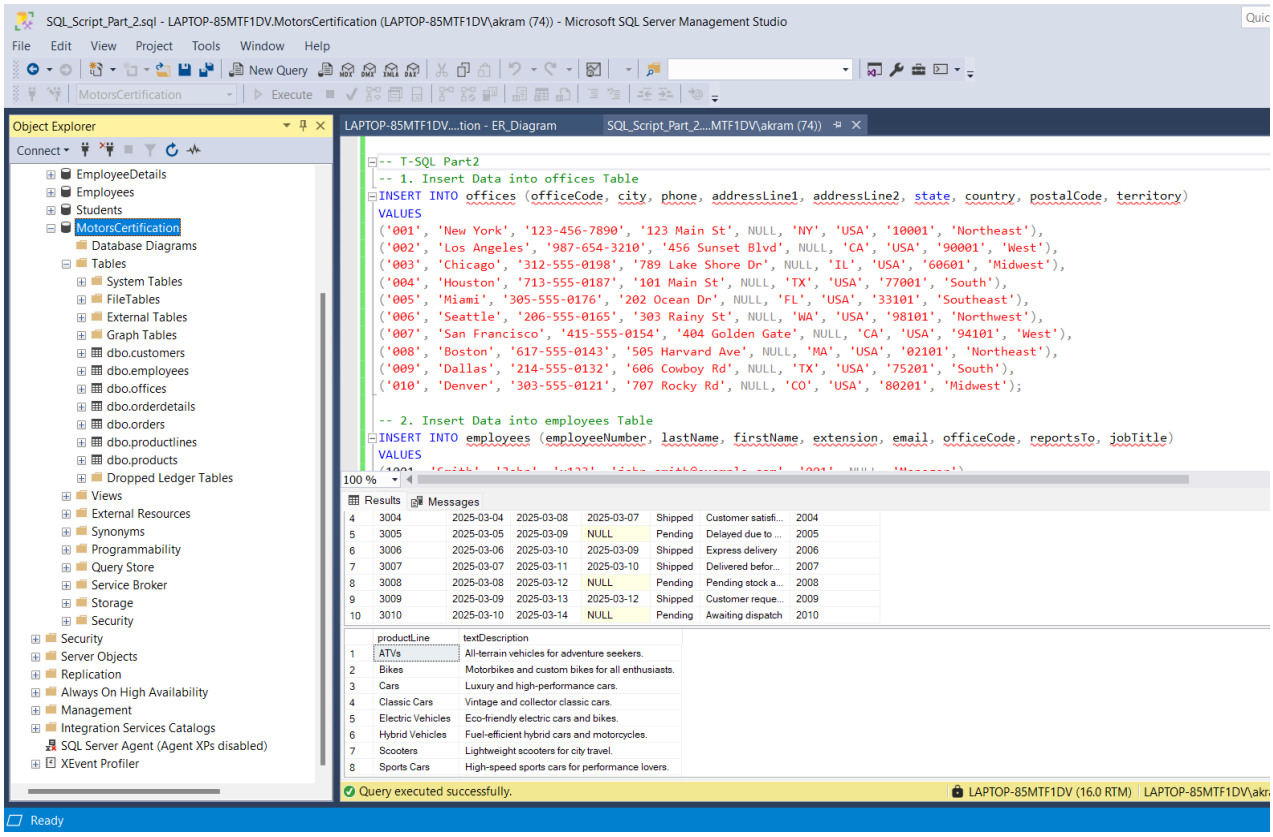
Tables creation



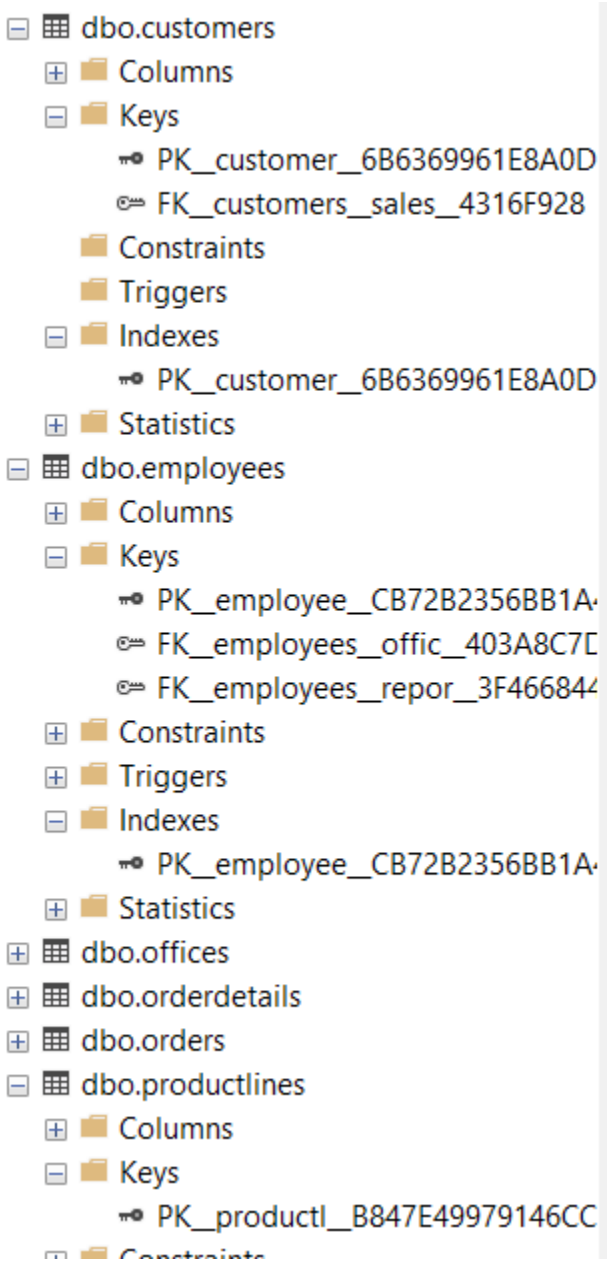
Check Database and table creation



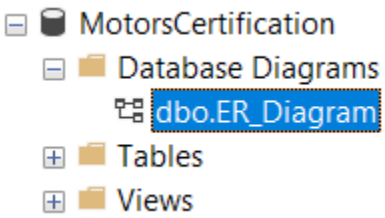
Data insertion

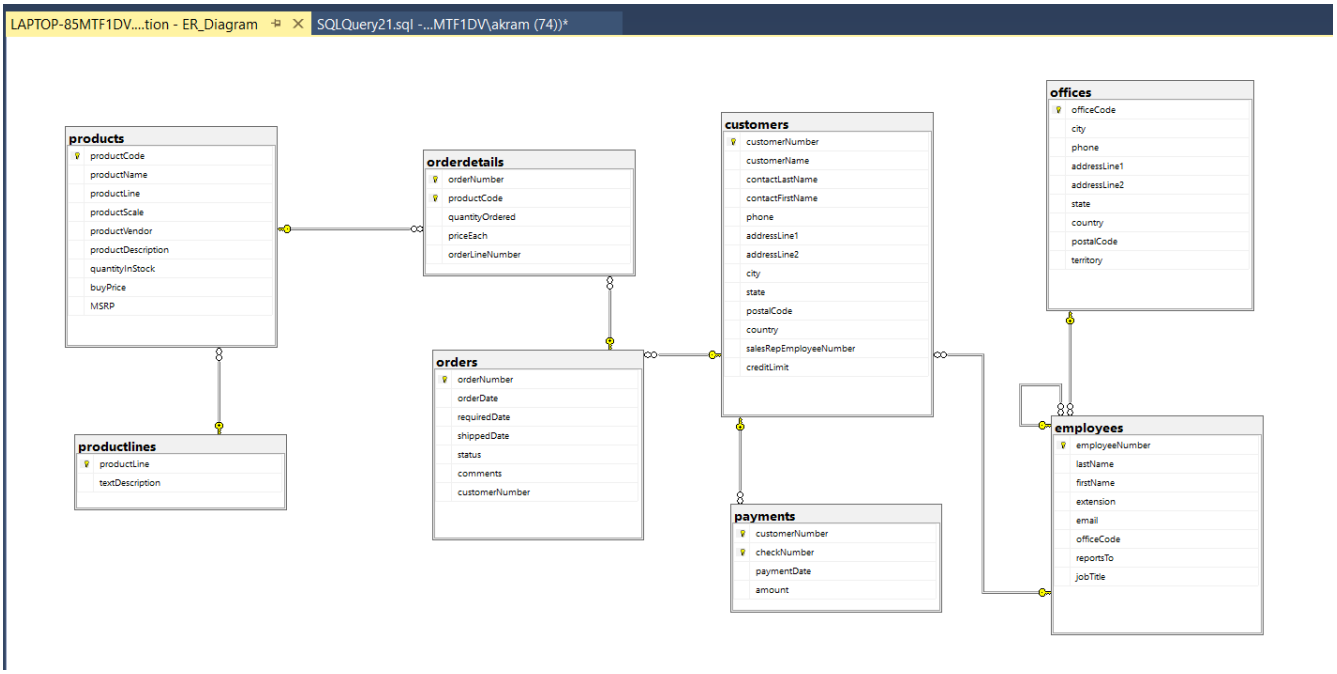


Check Primary Foreign Keys and Indexes



ER Diagram


MotorsCertification
Database Diagrams
dbo.ER_Diagram
Tables
Views


LAPTOP-85MTF1DV....tion - ER_Diagram
SQLQuery21.sql -...MTF1DV\akram (74))*
products
productCode
productName
productLine
productScale
productVendor
productDescription
quantityInStock
buyPrice
MSRP
orderdetails
orderNumber
productCode
quantityOrdered
priceEach
orderLineNumber
customers
customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit
offices
officeCode
city
phone
addressLine1
addressLine2
state
country
postalCode
territory
productlines
productLine
textDescription
orders
orderNumber
orderDate
requiredDate
shippedDate
status
comments
customerNumber
payments
customerNumber
checkNumber
paymentDate
amount
employees
employeeNumber
lastName
firstName
extension
email
officeCode
reportsTo
jobTitle

6 Find Out the Highest and Lowest Amount

SQL_Script_Part_3....MTF1DV\akram (52)) | SQL_Script_Projec...

```sql
USE MotorsCertification;

-- 6. Find Out the Highest and Lowest Amount
SELECT
    MAX(amount) AS HighestAmount,
    MIN(amount) AS LowestAmount
FROM payments;
```

100 %

Results | Messages

	HighestAmount	LowestAmount
1	120000	18000

7 Get the Unique Count of `customerName` from `customers`

```sql
-- 7. Get the Unique Count of customerName from customers
SELECT COUNT(DISTINCT customerName) AS UniqueCustomerCount FROM customers;
```

100 %

Results | Messages

	UniqueCustomerCount
1	10

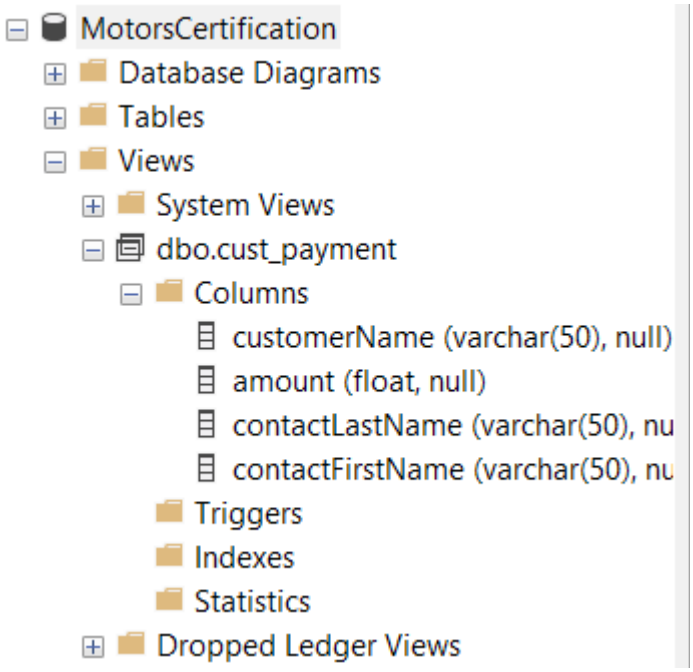
8 Create a View `cust_payment`

```sql
-- 8. Create a View cust_payment
CREATE VIEW cust_payment AS
SELECT
    c.customerName,
    p.amount,
    c.contactLastName,
    c.contactFirstName
FROM payments p
JOIN customers c ON p.customerNumber = c.customerNumber;

-- Select from the View
SELECT * FROM cust_payment;

-- Truncate and Drop the View
DROP VIEW cust_payment;
```

100 %

Results | Messages

	customerName	amount	contactLastName	contactFirstName
1	Hercules Garage	30000	Doe	Jane
2	Speedy Motors	18000	Johnson	Michael
3	Luxury Cars Ltd	75000	Brown	Emily
4	MotoHub	120000	Wilson	Daniel
5	Elite Auto	95000	Davis	Sarah
6	Turbo Cars	84000	Martinez	Laura
7	Superbike World	35000	Anderson	James
8	Hyper Cars	34000	Thomas	Patricia
9	Moto Enthusiasts	28000	Hernandez	Robert
10	HighSpeed Auto	60000	Gonzalez	David



9 Create a Stored Procedure for `Classic Cars` Product Line

This stored procedure will **display productLine for "Classic Cars"**.

```sql
-- 9. Create a Stored Procedure for Classic Cars Product Line
CREATE PROCEDURE GetClassicCars
AS
BEGIN
    SELECT * FROM products WHERE productLine = 'Classic Cars';
END;

-- Execute the Stored Procedure
EXEC GetClassicCars;
```

100 %

Results | Messages

	productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyPrice	MSRP
1	P007	Chevrolet Camaro	Classic Cars	1:18	Chevrolet	American muscle car.	20	35000	38000

10 Create a Function to Get `creditLimit` of Customers Less Than 96800

This function will **return a table of customers** with `creditLimit` **less than 96,800**.

```sql
-- 9. Create a Stored Procedure for Classic Cars Product Line
CREATE PROCEDURE GetClassicCars
AS
BEGIN
    SELECT * FROM products WHERE productLine = 'Classic Cars';
END;

-- Execute the Stored Procedure
EXEC GetClassicCars;
```

100 %

Results | Messages

	productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyPrice	MSRP
1	P007	Chevrolet Camaro	Classic Cars	1:18	Chevrolet	American muscle car.	20	35000	38000

Difference Between Stored Procedure and Function

Feature	Stored Procedure	Function
Purpose	Performs an **action** (e.g., inserting, updating, or deleting data)	Returns **a value or table** (used for calculations or queries)
Return Type	Can return **multiple result sets** or **nothing**	Must return **a single value** or **a table**
Usage in Queries	Cannot be used inside `SELECT` statements	Can be used inside `SELECT`, `WHERE`, `JOIN`
Can Modify Data?	✅Yes (INSERT, UPDATE, DELETE)	❌No (Only SELECT is allowed)
Calling Method	`EXEC ProcedureName`	`SELECT * FROM FunctionName()`
Example Use Case	Retrieve product details, process orders, update stock	Calculate tax, filter customers by credit limit

11 Create a Trigger to Store Transaction Record for Employee Table

This trigger will store a transaction record whenever a **new employee** is inserted. It will log `employeeNumber`, `lastName`, `firstName`, and `officeCode`.

SQL_Script_Part_3....MTF1DV\akram (52))* | SQL_Script_Projec...MTF1DV\akram (57)) | LAPTOP-85MTF1DV....tion - ER_Diagram | SQL_Script

```sql
-- 11. Create a Trigger to Store Transaction Record for Employee Table
-- Step 1: Create an Audit Table
CREATE TABLE EmployeeAudit (
    auditID INT IDENTITY(1,1) PRIMARY KEY,
    employeeNumber INT,
    lastName VARCHAR(50),
    firstName VARCHAR(50),
    officeCode VARCHAR(50),
    insertedAt DATETIME DEFAULT GETDATE()
);

-- Step 2: Create the Trigger
CREATE TRIGGER trg_Employee_Insert
ON employees
AFTER INSERT
AS
BEGIN
    INSERT INTO EmployeeAudit (employeeNumber, lastName, firstName, officeCode)
    SELECT employeeNumber, lastName, firstName, officeCode FROM inserted;

    PRINT 'Employee transaction recorded successfully.';
END;

-- Insert a New Employee (Trigger Will Fire Automatically)
INSERT INTO employees (employeeNumber, lastName, firstName, extension, email, officeCode, reportsTo, jobTitle)
VALUES (1011, 'Williams', 'Sophia', 'x133', 'sophia.williams@example.com', '003', 1001, 'Sales Rep');
-- Check If the Trigger Stored the Transaction in EmployeeAudit
SELECT * FROM EmployeeAudit;
```

100 %

Messages

```
(1 row affected)
Employee transaction recorded successfully.

(1 row affected)

Completion time: 2025-03-07T23:43:55.8026852+01:00
```

100 %

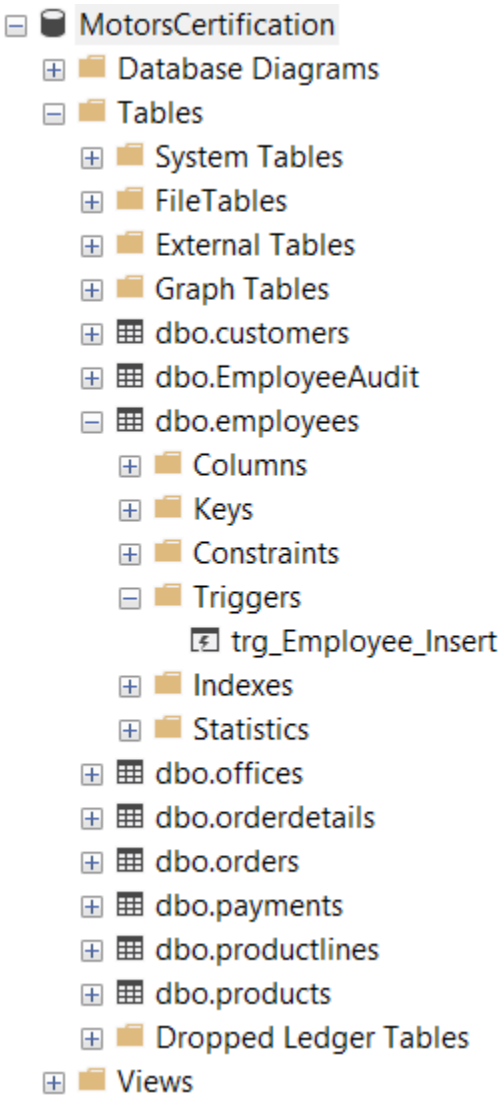
Query executed successfully. | LAPTOP-85MTF1DV (16.0 RTM) | LAPTOP-8

```sql
-- Insert a New Employee (Trigger Will Fire Automatically)
INSERT INTO employees (employeeNumber, lastName, firstName, extension, email, officeCode, reportsTo, jobTitle)
VALUES (1011, 'Williams', 'Sophia', 'x133', 'sophia.williams@example.com', '003', 1001, 'Sales Rep');
-- Check If the Trigger Stored the Transaction in EmployeeAudit
SELECT * FROM EmployeeAudit;
```

100 %

Results | Messages

	auditID	employeeNumber	lastName	firstName	officeCode	insertedAt
1	1	1011	Williams	Sophia	003	2025-03-07 23:43:55.797


MotorsCertification
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.customers
dbo.EmployeeAudit
dbo.employees
Columns
Keys
Constraints
Triggers
trg_Employee_Insert
Indexes
Statistics
dbo.offices
dbo.orderdetails
dbo.orders
dbo.payments
dbo.productlines
dbo.products
Dropped Ledger Tables
Views

12 Create a Trigger to Display Customer Number If Amount > 10,000

Whenever a new **payment** is added, if the `amount` is greater than **10,000**, the trigger will print the `customerNumber`.

```sql
-- Whenever a new payment is added, if the amount is greater than 10,000, the trigger will
CREATE TRIGGER trg_HighPayment
ON payments
AFTER INSERT
AS
BEGIN
    DECLARE @customerNumber INT;

    SELECT @customerNumber = customerNumber FROM inserted WHERE amount > 10000;

    IF @customerNumber IS NOT NULL
    BEGIN
        PRINT 'Customer Number with High Payment: ' + CAST(@customerNumber AS VARCHAR);
    END
END;

-- Insert a Payment Greater Than 10,000 (Trigger Should Fire)
INSERT INTO payments (customerNumber, checkNumber, paymentDate, amount)
VALUES (2005, 'CHK011', '2025-03-12', 9500.00); -- Amount < 10,000
INSERT INTO payments (customerNumber, checkNumber, paymentDate, amount)
VALUES (2006, 'CHK012', '2025-03-13', 15000.00); -- Amount > 10,000

SELECT * FROM sys.triggers WHERE name = 'trg_HighPayment';
```

```
Messages
Customer Number with High Payment: 2006

(1 row affected)

Completion time: 2025-03-07T23:53:04.8894180+01:00
```

Insert a Payment Greater Than 10,000 (Trigger Should Fire)

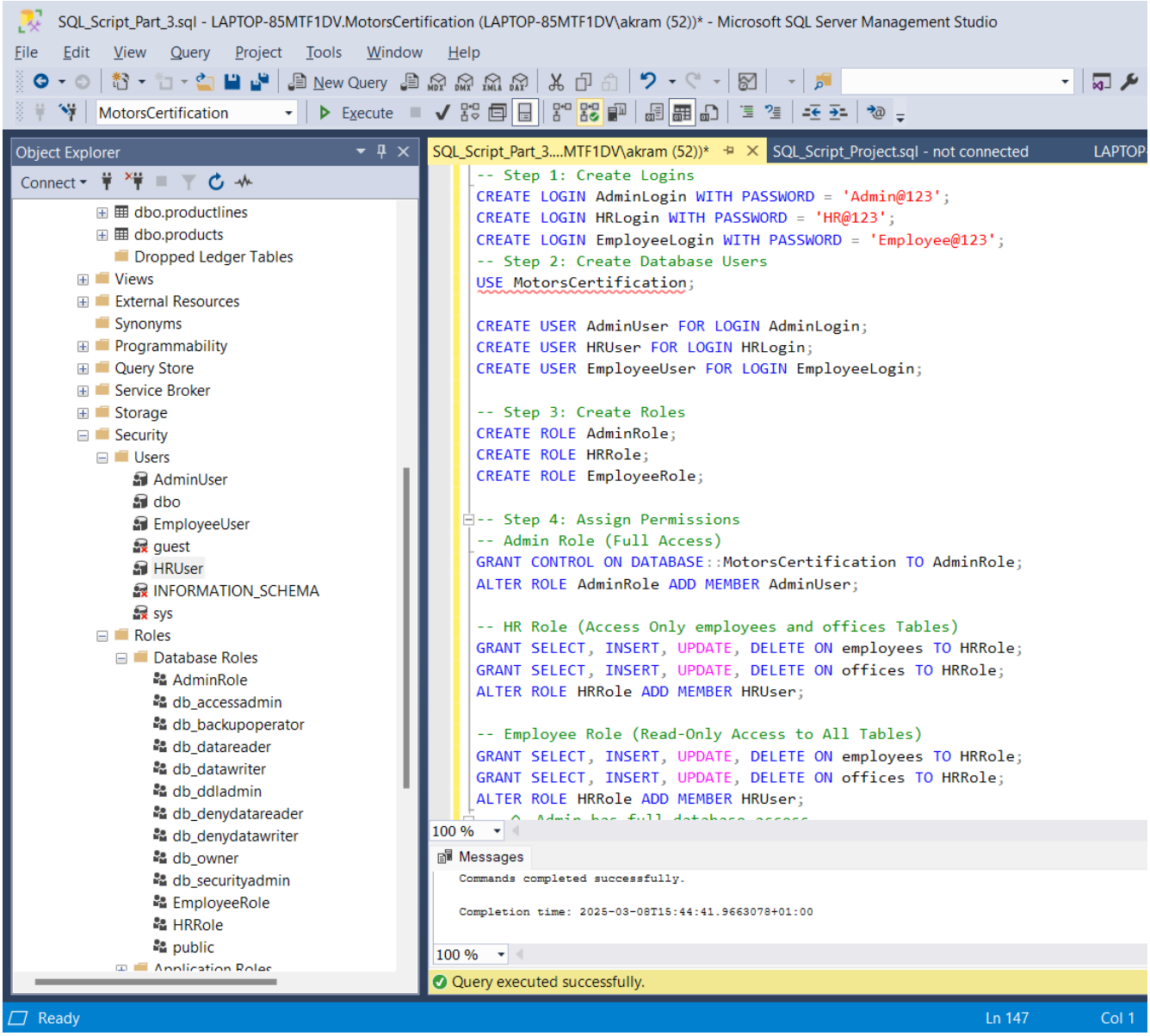
```sql
SELECT * FROM sys.triggers WHERE name = 'trg_HighPayment';
```

Results | Messages

	name	object_id	parent_class	parent_class_desc	parent_id	type	type_desc	create_date
1	trg_HighPayment	1669580986	1	OBJECT_OR_COLUMN	1333579789	TR	SQL_TRIGGER	2025-03-07 23:51:17.197

13 Create Users, Roles, and Logins

Step 1: Create Logins
Step 2: Create Database Users
Step 3: Create Roles
Step 4: Assign Permissions



- Security
  - Users
    - AdminUser
    - dbo
    - EmployeeUser
    - guest
    - HRUser
    - INFORMATION_SCHEMA
    - sys
  - Roles
    - Database Roles
      - AdminRole
      - db_accessadmin
      - db_backupoperator
      - db_datareader
      - db_datawriter
      - db_ddladmin
      - db_denydatareader
      - db_denydatawriter
      - db_owner
      - db_securityadmin
      - EmployeeRole
      - HRRole
      - public
    - Application Roles
  - Schemas

14 Schedule a Job for Database Backup

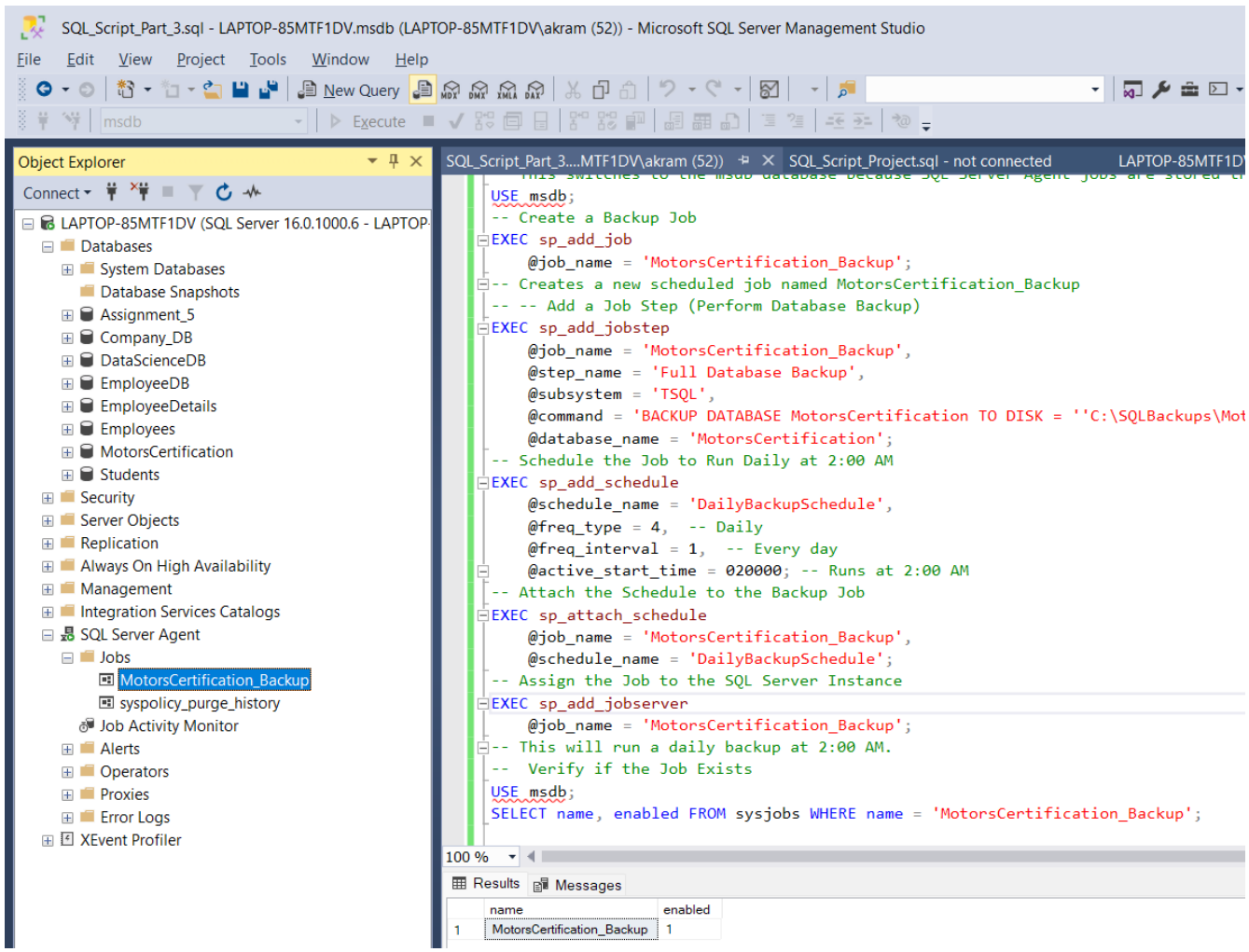
This job will **automatically back up the database** on a scheduled basis.

Msdb in SQL Server?

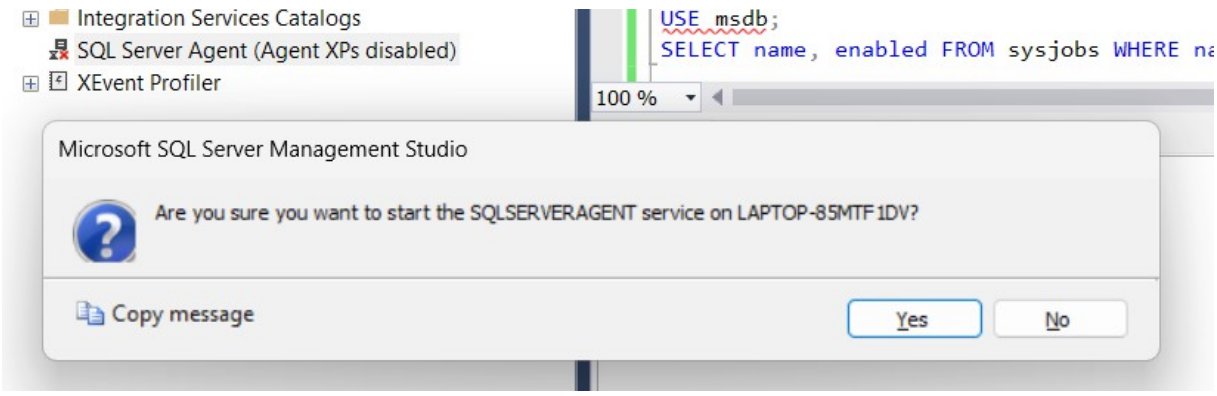
`msdb` is a **system database** in **SQL Server** that is used by **SQL Server Agent** for **scheduling jobs, alerts, and backups**.

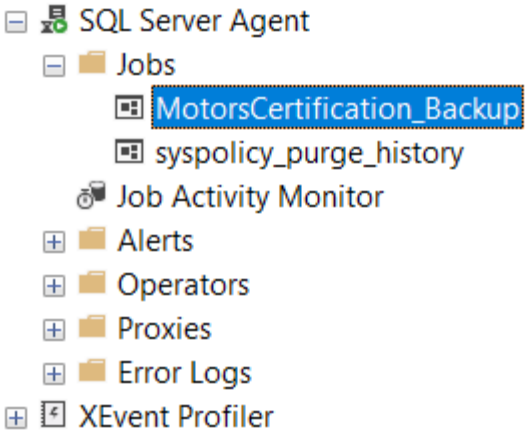
We are creating a **SQL Server Agent Job** to automatically **back up the database**. SQL Server Agent uses `msdb` to **store job-related information** such as:

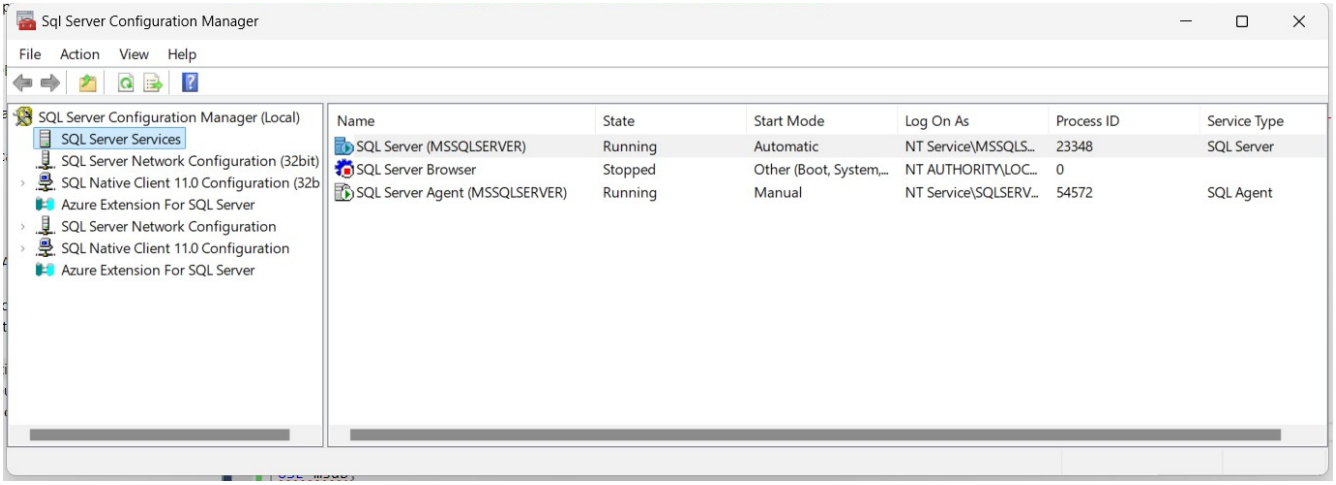
- **Scheduled Jobs**
- **Backup History**
- **Alerts & Notifications**
- **Job Steps and Execution History**



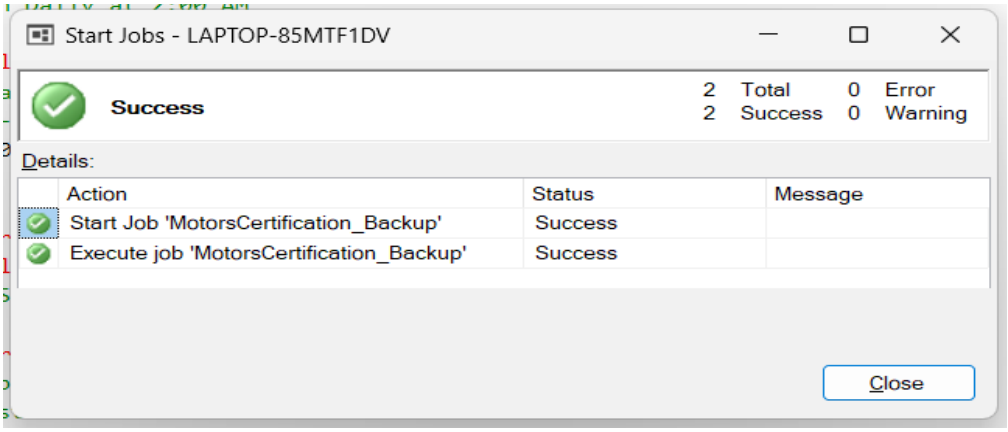
Enable SQL Server Agent



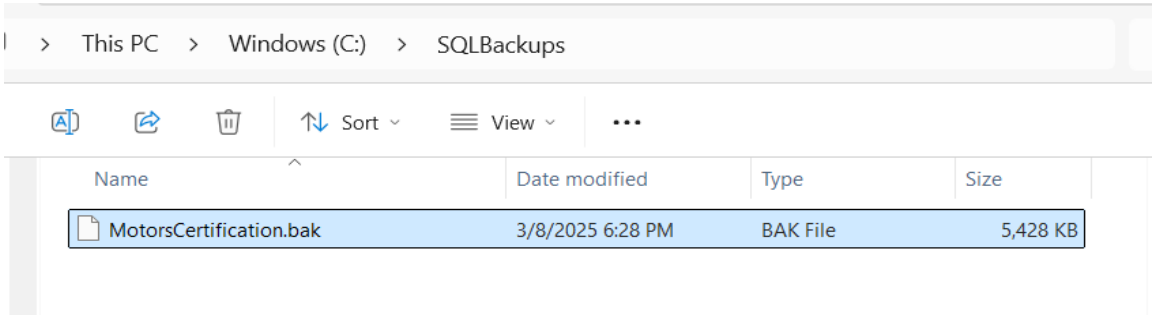


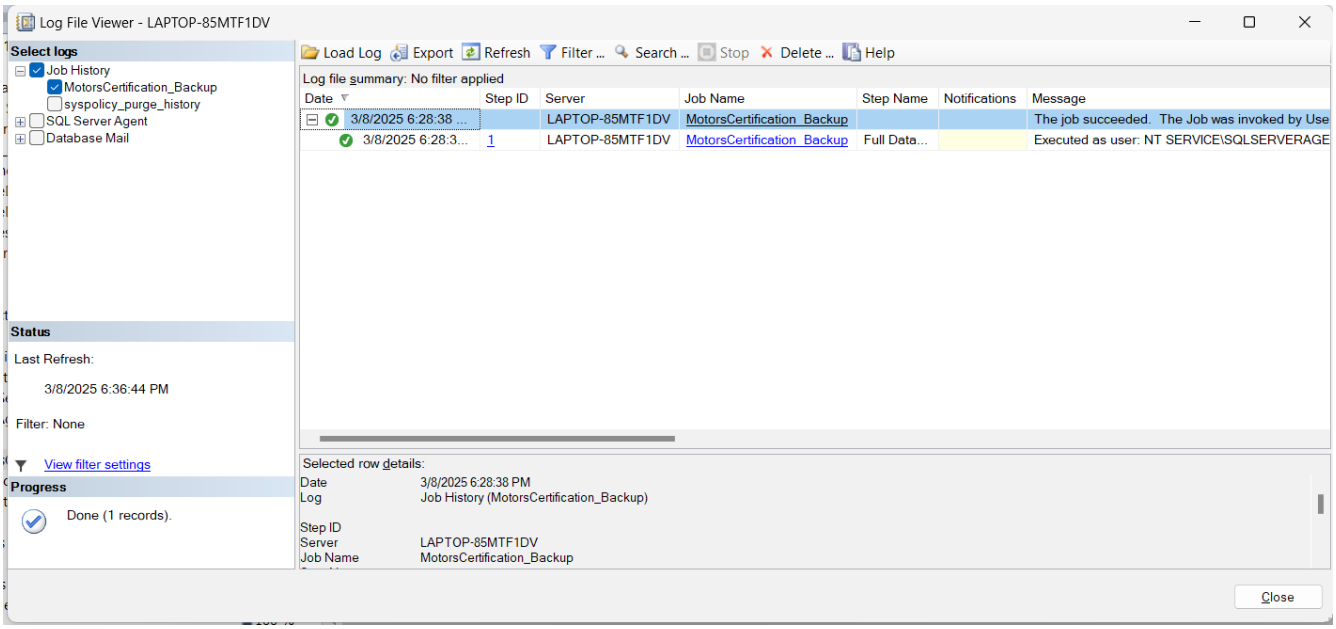


Manually Run the Job to test



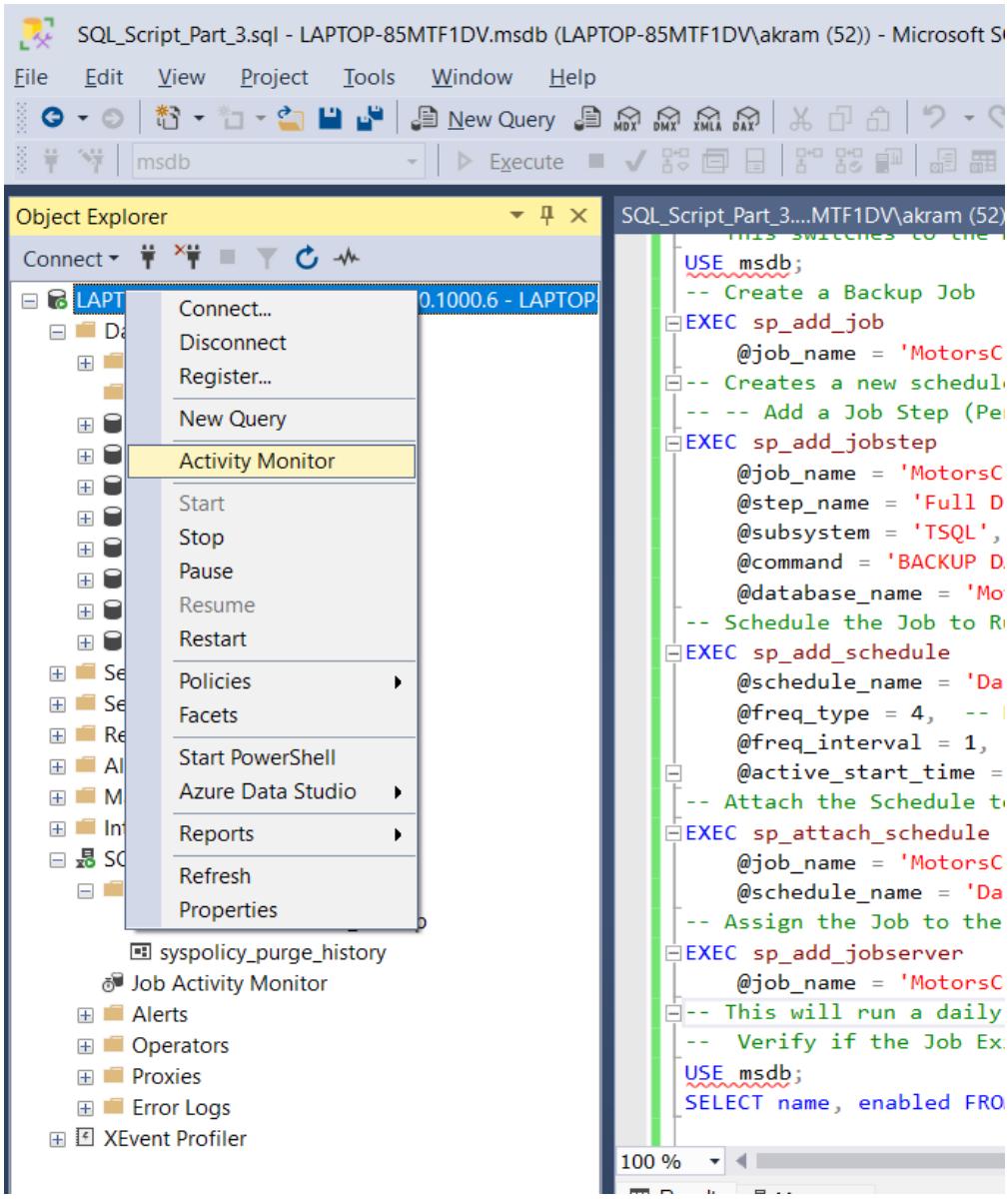
Verify If the Backup Was Created





15. Open Activity Monitor & List Observations

Open Activity Monitor:

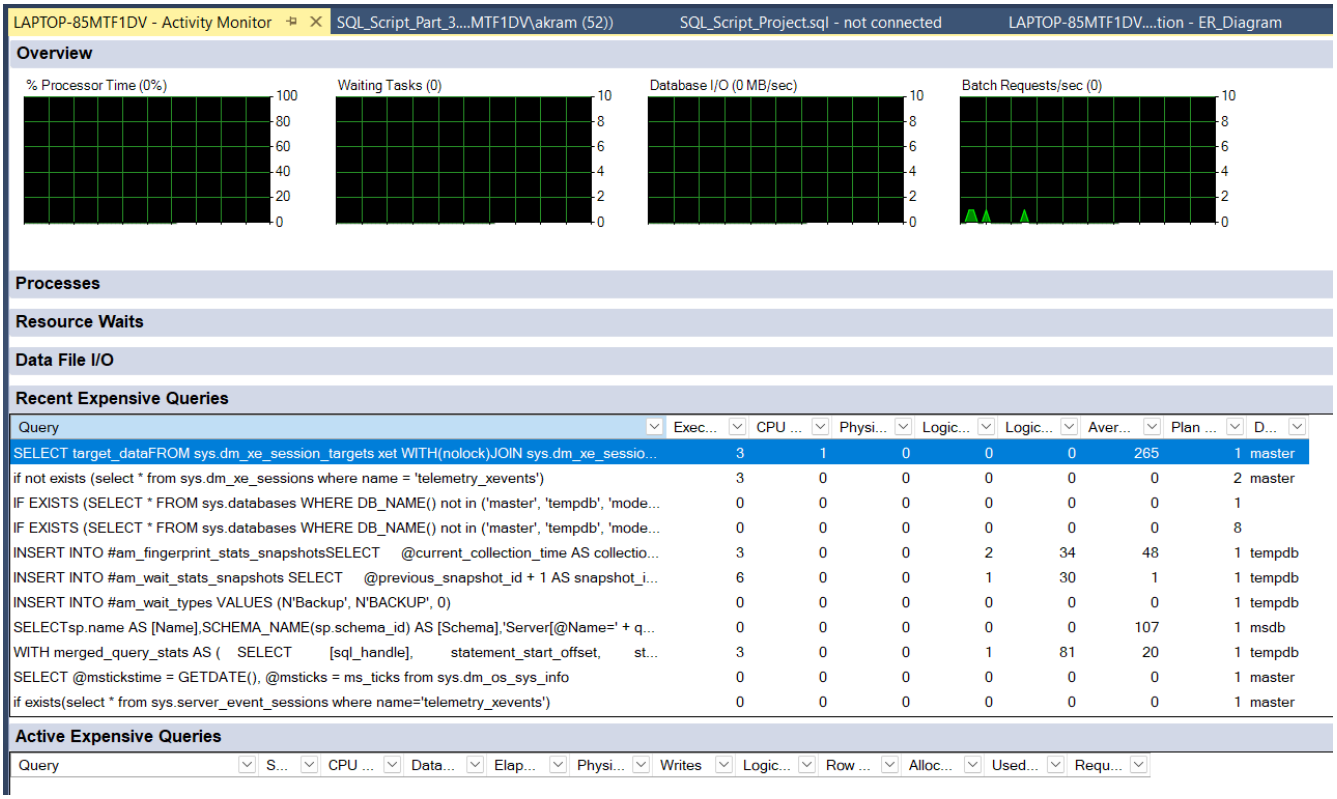


Observations in Activity Monitor:

- **Processes:**
  - Some background system processes are always running.
  - Query execution threads are active based on workload.
- **Resource Waits:**
  - No significant waits in a low-traffic environment.
  - Increased wait times may indicate locks or performance bottlenecks.
- **Active Expensive Queries:**

- Expensive queries usually involve **large joins** or **missing indexes**.
- **CPU-intensive queries** may indicate performance tuning is needed.

Recommendation:
Monitor **CPU usage**, **I/O waits**, and **blocking queries** for optimization.



16. Migrate SQL Server to Azure

Steps to Migrate SQL Server Database to Azure SQL Database

1. **Use Azure Data Migration Assistant (DMA)**
   - Download and install **Azure DMA**.
   - Analyze the database for **compatibility issues**.
   - Migrate schema and data.
2. **Use SQL Server Management Studio (SSMS)**
   - Right-click the **database → Tasks → Deploy Database to Microsoft Azure SQL Database**.
   - Provide **Azure SQL Server credentials**.
   - Complete the migration.
3. **Use Azure Database Migration Service (DMS)**
   - In **Azure Portal**, create a **Migration Service**.
   - Select **Online or Offline Migration**.
   - Provide **source and target database details**.
   - Start **data transfer**.