Database Design and Implementation for Driving School Operations: The EasyDrive System

Maharishi International University

Department of Computer Science

Fairfield, Iowa

Prepared By

- Bereket Fekadie ID=118384
- Abdullah al Zayed ID=618964

Date: 11/20/2025

Table of Contents

Acknowledgment

we would like to sincerely thank Prof. Mrudula Mukadam for her guidance, support, and valuable feedback throughout the CS422 – DBMS course project. Her insights and encouragement greatly helped us understand database concepts and successfully complete this project.

Introduction

This project presents the EasyDrive Management System, a comprehensive relational database solution designed to streamline and optimize the operations of a multi-location driving school network. Developed as an academic project at Maharishi International University's Department of Computer Science, the system addresses the complex data management needs of modern driving schools by integrating critical operational components including office management, staff coordination, vehicle tracking, client registration, lesson scheduling, appointment handling, driving test records, and vehicle inspections.

The database architecture adheres to the principles of normalization to ensure data integrity and reduce data redundancy, while it also contains strategic indexing for improved performance in queries. Through its interconnected tables and carefully designed relationships, the EasyDrive system provides a robust foundation for managing the entire lifecycle of driving instruction-from initial client interviews and lesson bookings to test preparation and vehicle maintenance-enabling driving school administrators and instructors to efficiently track student progress, manage resources, and maintain compliance with safety and licensing requirements across multiple office locations. This project is divided into two: part 1(mainly ER diagram, logical database) and part 2 (view, cursor, user defined function, trigger and store procedure).

A. PROJECT PART 1

ER Diagram: A visual representation of entities, attributes, and relationships in a system. Purpose: To model real-world data conceptually and understand how data connects.

Logical Database Design: Converts the ER diagram into tables, keys, and constraints for a specific DBMS. Purpose: To define the structure of the database in a way that can be implemented in SQL. In short: ER = conceptual map, Logical DB Design = blueprint for actual database tables.

1. Entity relationship Diagram

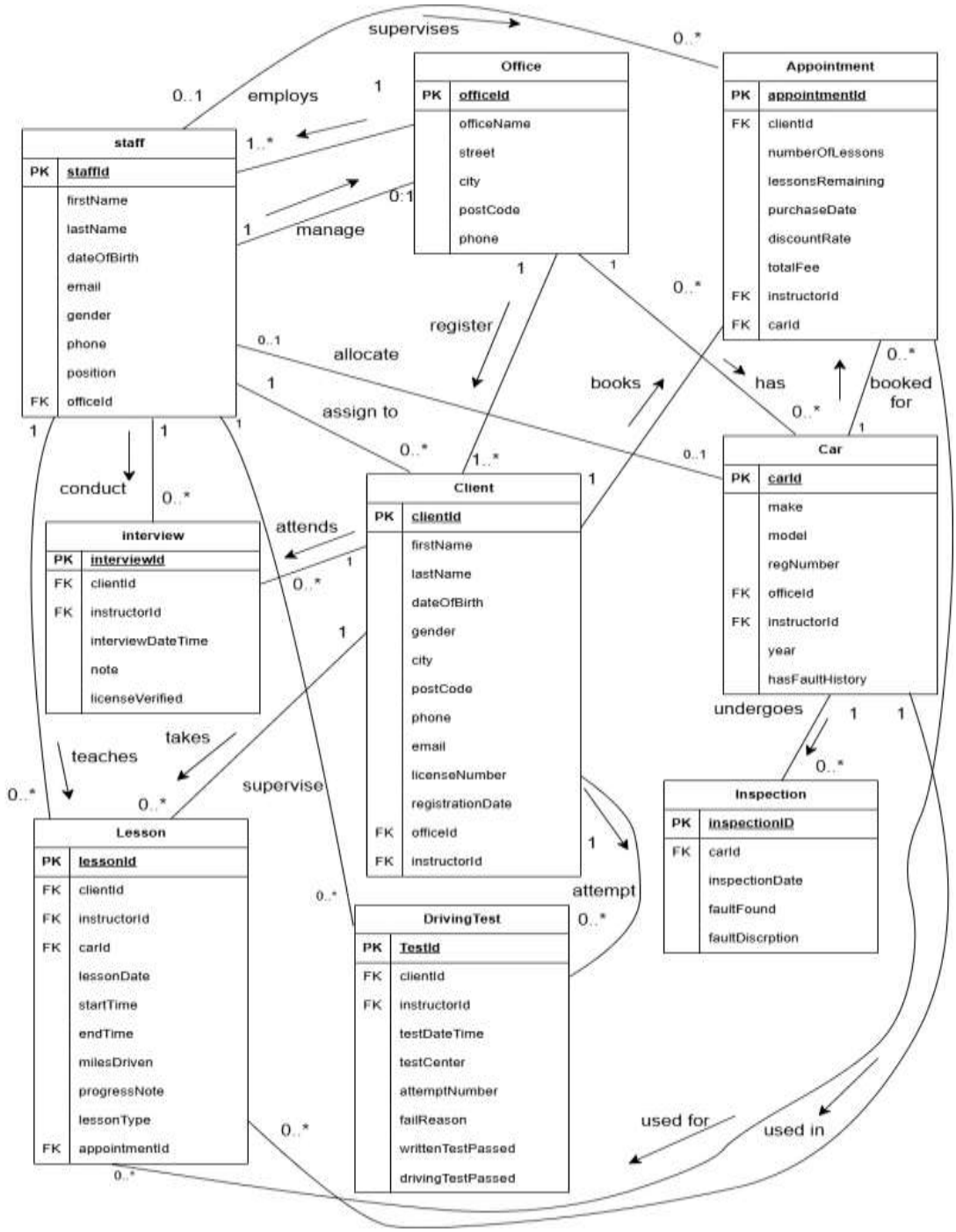


Figure 1ER- Diagram

2. Logical Database Design

OfficeStaff(officeId(pk), staffId(ak), firstName, lastName, dateOfBirth, email, gender, phone, position, officeName, street, city, postCode)

Staff(staffId(pk), carId(fk), officeId(fk), firstName, lastName, dateOfBirth, email, gender, phone, position)

StaffCar({staffId,carId } pk)

Office(officeId(pk), officeName, street, city, postCode, phone)

Appointment(appointmentId(pk), clientId(fk), instructorId(fk), carId(fk), numberOfLessons, lessonsRemaining,purchaseDate,discountRate,totalFee)

Client(clientId(pk), officeId, instructorId(fk), firstName, lastName, dateOfBirth, gender, city, postCode, phone, email, licenseNumber, registrationDate)

Car (carId (pk), officeId(fk), instructorId(fk), make, model, year, regNumber, hasFaultHistory)

Inspection(inspectionId(pk), carId, inspectionDate, faultFound, faultDescription)

DrivingTest(testId(pk), clientId(fk), instructorId(fk), testDateTime, testCenter, drivingTestPassed, writtenTestPassed, attemptNumber, failReason)

Lesson(lessonId(pk), clientId(fk), instructorId(fk), carId(fk), appointmentId(fk), lessonDate, startTime, endTime, milesDriven, progressNote)

Interview(interviewId(pk), clientId(fk), instructorId(fk), interviewDateTime, notes, licenseVerified)

3. Normalized Table

1. Office(officeId(pk), officeName, street, city, postCode, phone)

2. Staff(staffId(pk), officeId(fk), firstName, lastName, dateOfBirth,
email, gender, phone, position, supervisorId(fk))

3. Car(carId(pk), officeId(fk), instructorId(fk), make, model, year,
regNumber, lastInspectionDate, hasFaultHistory)

4. Client(clientId(pk), officeId(fk), instructorId(fk), firstName, lastName,
dateOfBirth, gender, city, postCode, phone, email,
licenseNumber, registrationDate)

5. Interview(interviewId(pk), clientId(fk), instructorId(fk),
interviewDateTime, notes, licenseVerified)

6. Appointment(appointmentId(pk), clientId(fk), instructorId(fk), carId(fk),
numberOfLessons, lessonsRemaining, purchaseDate,
discountRate, totalFee)

7. Lesson(lessonId(pk), clientId(fk), instructorId(fk), carId(fk),
appointmentId(fk), lessonDate, startTime, endTime,
milesDriven, progressNote, lessonType)

8. DrivingTest(testId(pk), clientId(fk), instructorId(fk), testDateTime,
testCenter, drivingTestPassed, writtenTestPassed,
attemptNumber, failReason)

9. Inspection(inspectionId(pk), carId(fk), inspectionDate,
faultFound, faultDescription)

4. Creating Project Data Base and Its Table

```
CREATE DATABASE projectDB;
GO
USE projectDB;
GO
```

1) OFFICE

```
CREATE TABLE dbo.Office (
 officeId   INT IDENTITY (1,1) PRIMARY KEY,
 officeName VARCHAR (100) NOT NULL,
 Street VARCHAR (100) NULL,
 city      VARCHAR (50) NOT NULL,
 postCode   VARCHAR (10) NULL,
 phone     VARCHAR (20) NULL
);
```

2) STAFF

```
CREATE TABLE dbo.Staff (
 staffId     INT IDENTITY (1,1) PRIMARY KEY,
 officeId    INT NOT NULL,
 firstName   VARCHAR (50) NOT NULL,
 lastName    VARCHAR (50) NOT NULL,
 dateOfBirth DATE NULL,
 email       VARCHAR (100) NULL,
 gender      CHAR (1) NOT NULL
          CONSTRAINT CK_Staff_Gender CHECK (gender IN ('M','F')),
 phone       VARCHAR (20) NULL,
 position    VARCHAR (50) NOT NULL,
 supervisorId INT NULL,

 CONSTRAINT FK_Staff_Office    FOREIGN KEY (officeId) REFERENCES dbo.Office(officeId),
 CONSTRAINT FK_Staff_Supervisor FOREIGN KEY (supervisorId) REFERENCES dbo.Staff(staffId)
);
```

3) CAR

```
CREATE TABLE dbo.Car (
 carId          INT IDENTITY (1,1) PRIMARY KEY,
 officeId       INT NOT NULL,
 instructorId    INT NULL,
 make           VARCHAR (50) NOT NULL,
 model          VARCHAR (50) NOT NULL,
 year           INT NULL,
 regNumber      VARCHAR (20) NOT NULL UNIQUE,
 lastInspectionDate DATE NULL,
 hasFaultHistory   BIT NOT NULL DEFAULT (0),
 CONSTRAINT FK_Car_Office    FOREIGN KEY (officeId) REFERENCES dbo.Office(officeId),
 CONSTRAINT FK_Car_Instructor FOREIGN KEY (instructorId) REFERENCES dbo.Staff(staffId)
);
```

4) CLIENT

```
CREATE TABLE dbo.Client (
 clientId      INT IDENTITY (1,1) PRIMARY KEY,
 officeId      INT NOT NULL,
 instructorId   INT NULL,
 firstName     VARCHAR (50) NOT NULL,
 lastName      VARCHAR (50) NOT NULL,
 dateOfBirth    DATE NULL,
```

```
 gender        CHAR (1) NOT NULL
               CONSTRAINT CK_Client_Gender CHECK (gender IN ('M', 'F')),
 city          VARCHAR (50) NULL,
 postCode      VARCHAR (10) NULL,
 phone         VARCHAR (20) NULL,
 email         VARCHAR (100) NULL,
 licenseNumber   VARCHAR (30) NULL,
 registrationDate DATE NOT NULL DEFAULT (GETDATE ()),
 CONSTRAINT FK_Client_Office    FOREIGN KEY (officeId) REFERENCES dbo.Office(officeId),
 CONSTRAINT FK_Client_Instructor FOREIGN KEY (instructorId) REFERENCES dbo.Staff(staffId)
);
```

5) INTERVIEW

```
CREATE TABLE dbo.Interview (
 interviewId      INT IDENTITY (1,1) PRIMARY KEY,
 clientId         INT NOT NULL,
 instructorId     INT NOT NULL,
 interviewDateTime DATETIME2(0) NOT NULL,
 notes            VARCHAR (500) NULL,
 licenseVerified   BIT NOT NULL DEFAULT (0),
 CONSTRAINT FK_Interview_Client    FOREIGN KEY (clientId) REFERENCES dbo.Client(clientId),
 CONSTRAINT FK_Interview_Instructor FOREIGN KEY (instructorId) REFERENCES dbo.Staff(staffId)
);
```

6) APPOINTMENT

```
CREATE TABLE dbo.Appointment (
 appointmentId    INT IDENTITY (1,1) PRIMARY KEY,
 clientId         INT NOT NULL,
 instructorId     INT NOT NULL,
 carId            INT NOT NULL,
 numberOfLessons INT NOT NULL CHECK (numberOfLessons > 0),
 lessonsRemaining INT NOT NULL CHECK (lessonsRemaining >= 0),
 purchaseDate     DATE NOT NULL,
 discountRate     DECIMAL (5,2) NULL,
 totalFee         DECIMAL (10,2) NOT NULL,
 CONSTRAINT FK_Appt_Client     FOREIGN KEY (clientId) REFERENCES dbo.Client(clientId),
 CONSTRAINT FK_Appt_Instructor FOREIGN KEY (instructorId) REFERENCES dbo.Staff(staffId),
 CONSTRAINT FK_Appt_Car        FOREIGN KEY (carId) REFERENCES dbo.Car(carId)
);
```

7) LESSON

```
CREATE TABLE dbo.Lesson (
 lessonId     INT IDENTITY (1,1) PRIMARY KEY,
 clientId     INT NOT NULL,
 instructorId INT NOT NULL,
 carId        INT NOT NULL,
 appointmentId INT NULL,
 lessonDate    DATE NOT NULL,
 startTime     TIME (0) NOT NULL,
 endTime       TIME (0) NOT NULL,
 milesDriven   DECIMAL (6,2) NULL,
 progressNote VARCHAR (500) NULL,
 lessonType    VARCHAR (20) NULL,
 CONSTRAINT CK_Lesson_TimeOrder CHECK (startTime < endTime),
 CONSTRAINT FK_Lesson_Client     FOREIGN KEY (clientId) REFERENCES dbo.Client(clientId),
 CONSTRAINT FK_Lesson_Instructor FOREIGN KEY (instructorId) REFERENCES dbo.Staff(staffId),
 CONSTRAINT FK_Lesson_Car        FOREIGN KEY (carId) REFERENCES dbo.Car(carId),
 CONSTRAINT FK_Lesson_Appointment FOREIGN KEY (appointmentId) REFERENCES
dbo.Appointment(appointmentId)
);
```

8) DRIVINGTEST

```
CREATE TABLE dbo.DrivingTest (
 testId          INT IDENTITY (1,1) PRIMARY KEY,
 clientId        INT NOT NULL,
 instructorId     INT NOT NULL,
 testDateTime     DATETIME2(0) NOT NULL,
 testCenter       VARCHAR (100) NOT NULL,
 drivingTestPassed BIT NOT NULL,
 writtenTestPassed BIT NOT NULL,
 attemptNumber    INT NOT NULL CHECK (attemptNumber > 0),
 failReason       VARCHAR (500) NULL,
 CONSTRAINT FK_Test_Client    FOREIGN KEY (clientId) REFERENCES dbo.Client(clientId),
 CONSTRAINT FK_Test_Instructor FOREIGN KEY (instructorId) REFERENCES dbo.Staff(staffId)
);
```

9) INSPECTION

```
CREATE TABLE dbo.Inspection (
 inspectionId    INT IDENTITY (1,1) PRIMARY KEY,
```

```
 carId         INT NOT NULL,
 inspectionDate DATE NOT NULL,
 faultFound     BIT NOT NULL DEFAULT (0),
 faultDescription VARCHAR (500) NULL,
 CONSTRAINT FK_Inspection_Car FOREIGN KEY (carId) REFERENCES dbo.Car(carId)
);
```

10) Indexes for fast retrieval and data integrity enhancement

```
CREATE INDEX IX_Staff_Office ON dbo.Staff(officeId);
CREATE INDEX IX_Staff_Supervisor ON dbo.Staff(supervisorId);
CREATE INDEX IX_Client_Office ON dbo.Client(officeId);
CREATE INDEX IX_Client_Instructor ON dbo.Client(instructorId);
CREATE INDEX IX_Car_Office ON dbo.Car(officeId);
CREATE INDEX IX_Car_Instructor ON dbo.Car(instructorId);
CREATE INDEX IX_Lesson_InstrDate ON dbo.Lesson(instructorId, lessonDate);
CREATE INDEX IX_Appt_InstrDate ON dbo.Appointment(instructorId, purchaseDate);
CREATE INDEX IX_Test_ClientAttempt ON dbo.DrivingTest(clientId, attemptNumber);
CREATE INDEX IX_Inspection_CarDate ON dbo.Inspection(carId, inspectionDate);
```

5. Seed Data

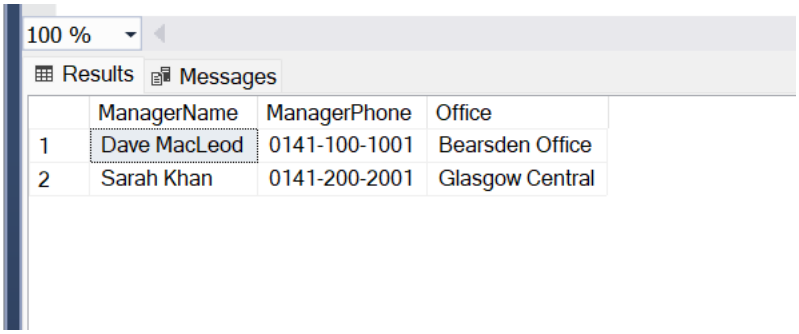
```
-- Offices
INSERT INTO dbo.Office (officeName, street, city, postCode, phone) VALUES
('Bearsden Office',     '12 Main St',    'Glasgow',  'G61 1AA', '0141-100-1000'),
('Glasgow Central',     '200 Queen St',   'Glasgow',  'G1 3AA',  '0141-200-2000'),
('Dundee Office',       '5 Highgate',    'Dundee',   'DD1 2BB', '0138-200-2222'),
('Aberdeen Office',     '77 Beach Blvd',  'Aberdeen', 'AB1 1CC', '0122-111-3333');

-- Staff
INSERT INTO dbo.Staff (officeId, firstName, lastName, dateOfBirth, email, gender, phone, position,
supervisorId) VALUES
(1, 'Dave',   'MacLeod', '1970-03-10','dave@easy.com','M','0141-100-1001','Manager', NULL),
(1, 'Anna',   'Brown',   '1983-05-20','anna@easy.com','F','0141-100-1002','Senior Instructor', 1),
(1, 'John',   'White',   '1985-07-12','john@easy.com','M','0141-100-1003','Instructor',       2),
(1, 'Emily',  'Clark',   '1990-09-22','emily@easy.com','F','0141-100-1004','Instructor',       2),
(1, 'Peter',  'Miles',   '1996-11-23','peter@easy.com','M','0141-100-1005','Admin',           1),
(2, 'Sarah',  'Khan',    '1975-02-14','sarah@easy.com','F','0141-200-2001','Manager',        NULL),
(2, 'Lewis',  'Grant',   '1965-08-01','lewis@easy.com','M','0141-200-2002','Instructor',      6),
(3, 'Megan',  'Ross',    '1988-01-18','megan@easy.com','F','0138-200-2223','Instructor',      NULL),
(4, 'Abdul',  'Hussain', '1992-12-02','abdul@easy.com','M','0122-111-3334','Instructor',      NULL);
```

```
-- Cars
INSERT INTO dbo.Car (officeId, instructorId, make, model, year, regNumber, lastInspectionDate,
hasFaultHistory) VALUES
(1, 3, 'Ford',   'Focus',  2021, 'GLA-123', '2025-10-01', 0),
(1, 4, 'Toyota', 'Yaris',  2022, 'GLA-555', '2025-10-05', 1),
(2, 7, 'VW',     'Polo',   2020, 'GLA-777', '2025-09-20', 0),
(3, 8, 'Honda',  'Jazz',   2019, 'DUN-222', '2025-08-10', 0);

-- Clients
INSERT INTO dbo.Client (officeId, instructorId, firstName, lastName, dateOfBirth, gender, city,
postCode, phone, email, licenseNumber, registrationDate) VALUES
(1, 4, 'Mary',  'Smith',  '2000-04-04','F','Glasgow','G61 1AA','07000-1111','mary@example.com','LIC-
001','2025-09-01'),
(1, 3, 'Peter', 'Jones',  '1999-02-02','M','Glasgow','G61 1AA','07000-2222','peter@example.com','LIC-
002','2025-09-05'),
(2, 7, 'Sara',  'Lopez',  '1998-11-30','F','Glasgow','G1 3AA','07000-3333','sara@example.com','LIC-
003','2025-09-10'),
(3, 8, 'Liam',  'Wood',   '2001-06-06','M','Dundee','DD1 2BB','07000-4444','liam@example.com','LIC-
004','2025-09-12'),
(1, 4, 'Nora',  'Evans',  '2002-08-18','F','Glasgow','G61 1AA','07000-5555','nora@example.com','LIC-
005','2025-09-20');

-- Interviews
INSERT INTO dbo.Interview (clientId, instructorId, interviewDateTime, notes, licenseVerified)
VALUES
(1, 4, '2025-09-02 10:00', 'Confident, basic control ok', 1),
(2, 3, '2025-09-06 11:30', 'Needs junction practice',     1),
(3, 7, '2025-09-11 09:00', 'License verified, some nerves',1),
(5, 4, '2025-09-21 15:30', 'New learner, prefers auto',   1);

-- Appointments
INSERT INTO dbo.Appointment (clientId, instructorId, carId, numberOfLessons, lessonsRemaining,
purchaseDate, discountRate, totalFee) VALUES
(1, 4, 2, 10, 8, '2025-10-01', 10.00, 350.00),
(2, 3, 1,  5,  3, '2025-10-03',  0.00, 200.00),
(3, 7, 3,  8,  8, '2025-10-05',  5.00, 280.00);


INSERT INTO dbo.Lesson (clientId, instructorId, carId, appointmentId, lessonDate, startTime,
endTime, milesDriven, progressNote, lessonType) VALUES
(1, 4, 2, 1, '2025-11-12', '09:00', '10:00', 12.5, 'Roundabouts', 'block'),
(1, 4, 2, 1, '2025-11-14', '14:00', '15:00', 10.0, 'Hill start',   'block'),
```

```
(2, 3, 1, 2, '2025-11-13', '11:00', '12:00', 11.2, 'Junctions',   'block'),
(3, 7, 3, 3, '2025-11-15', '10:00', '11:00',  9.8, 'City traffic', 'block'),
(2, 3, 1, 2, '2025-10-20', '10:00', '11:00',  8.0, 'Clutch control','block'),
(5, 4, 2, 1, '2025-10-21', '08:00', '09:00',  7.5, 'Mirrors',     'individual');

INSERT INTO dbo.DrivingTest (clientId, instructorId, testDateTime, testCenter, drivingTestPassed,
writtenTestPassed, attemptNumber, failReason) VALUES
(1, 4, '2013-01-15 10:00', 'Glasgow Center', 1, 1, 1, NULL),
(2, 3, '2025-10-01 09:00', 'Glasgow Center', 0, 1, 1, 'Observation'),
(2, 3, '2025-10-20 09:00', 'Glasgow Center', 0, 1, 2, 'Junctions'),
(2, 3, '2025-11-01 09:00', 'Glasgow Center', 0, 1, 3, 'Control'),
(2, 3, '2025-11-08 09:00', 'Glasgow Center', 0, 1, 4, 'Mirrors');

INSERT INTO dbo.Inspection (carId, inspectionDate, faultFound, faultDescription) VALUES
(1, '2025-06-01', 0, NULL),
(1, '2025-09-30', 0, NULL),
(2, '2025-07-10', 1, 'Brake light'),
(3, '2025-08-15', 0, NULL),
(4, '2025-08-20', 0, NULL);
```

6. SQL Query Execution

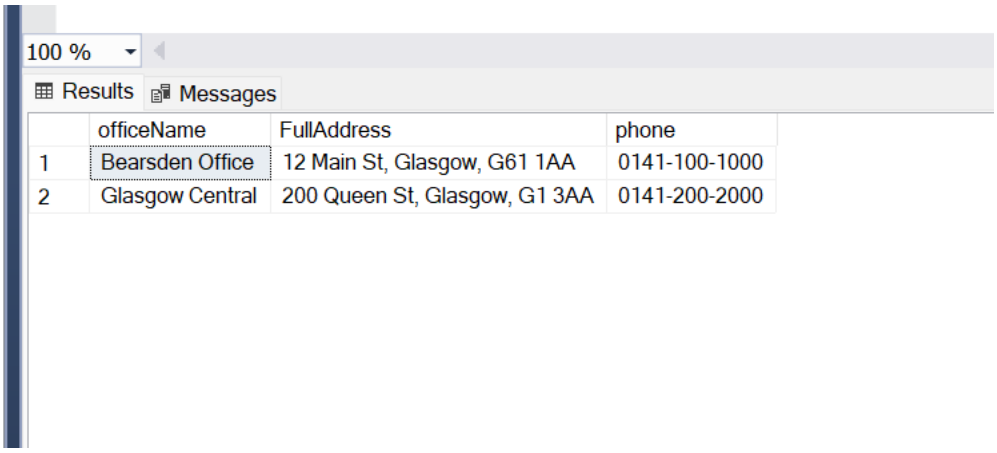
```
-- Query 1: The names and the telephone numbers of the Managers of each office
SELECT
  s.firstName + ' ' + s.lastName AS ManagerName,
  s.phone AS ManagerPhone,
  o.officeName AS Office
FROM Staff s
INNER JOIN Office o ON s.officeId = o.officeId
WHERE s.position = 'Manager'
ORDER BY o.officeName;
```



	ManagerName	ManagerPhone	Office
1	Dave MacLeod	0141-100-1001	Bearsden Office
2	Sarah Khan	0141-200-2001	Glasgow Central

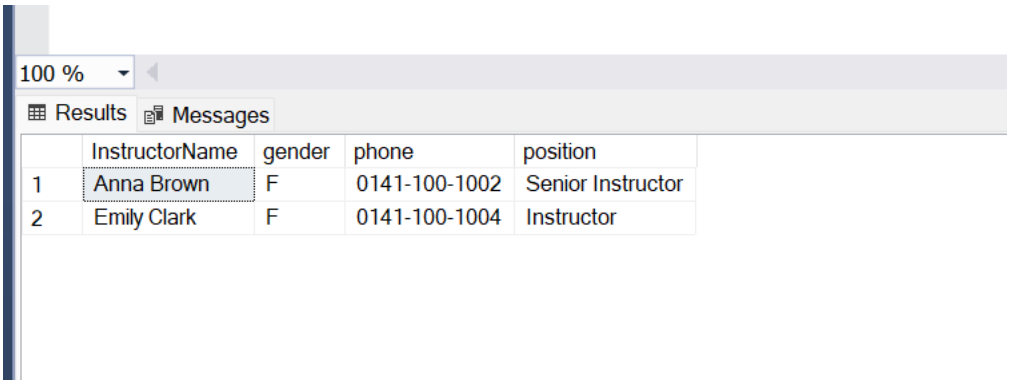
```
-- Query 2: The full address of all offices in Glasgow
SELECT
  officeName,
```

```
    street + ', ' + city + ', ' + postCode AS FullAddress,
    phone
FROM Office
WHERE city = 'Glasgow'
ORDER BY officeName;
```



	officeName	FullAddress	phone
1	Bearsden Office	12 Main St, Glasgow, G61 1AA	0141-100-1000
2	Glasgow Central	200 Queen St, Glasgow, G1 3AA	0141-200-2000

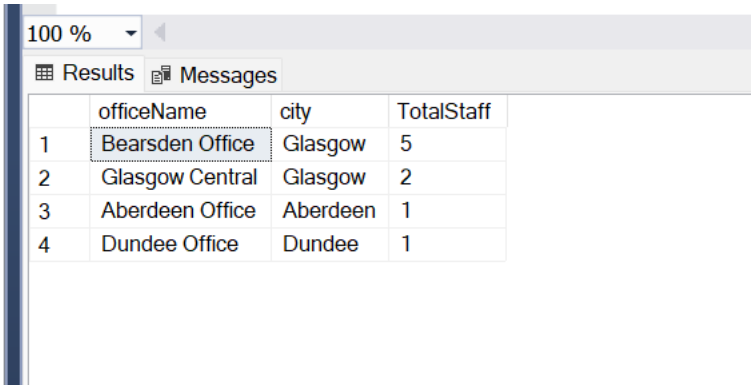
```
-- Query 3: The names of all female Instructors based in Glasgow, Bearsden office

SELECT
    s.firstName + ' ' + s.lastName AS InstructorName,
    s.gender,
    s.phone,
    s.position
FROM Staff s
INNER JOIN Office o ON s.officeId = o.officeId
WHERE s.gender = 'F'
  AND s.position IN ('Instructor', 'Senior Instructor')
  AND o.officeName = 'Bearsden Office'
ORDER BY s.lastName, s.firstName;
```



	InstructorName	gender	phone	position
1	Anna Brown	F	0141-100-1002	Senior Instructor
2	Emily Clark	F	0141-100-1004	Instructor

```
-- Query 4: The total number of staff at each office
SELECT
o.officeName,
```

```
o.city,
COUNT(s.staffId) AS TotalStaff
FROM Office o
LEFT JOIN Staff s ON o.officeId = s.officeId
GROUP BY o.officeId, o.officeName, o.city
ORDER BY TotalStaff DESC, o.officeName;
```

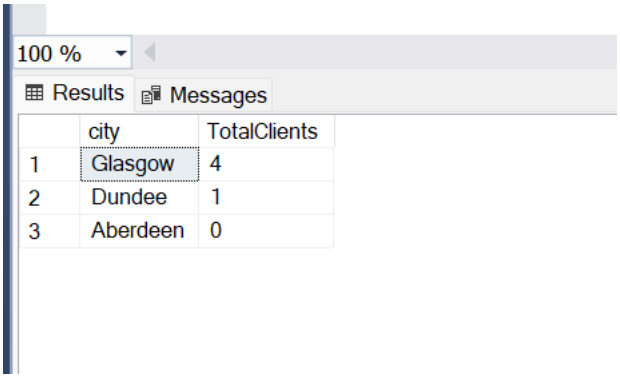


100 %

Results | Messages

	officeName	city	TotalStaff
1	Bearsden Office	Glasgow	5
2	Glasgow Central	Glasgow	2
3	Aberdeen Office	Aberdeen	1
4	Dundee Office	Dundee	1

```
-- Query 5: The total number of clients (past and present) in each city
SELECT
o.city,
COUNT(c.clientId) AS TotalClients
FROM Office o
LEFT JOIN Client c ON o.officeId = c.officeId
GROUP BY o.city
ORDER BY TotalClients DESC, o.city;
```



100 %

Results | Messages

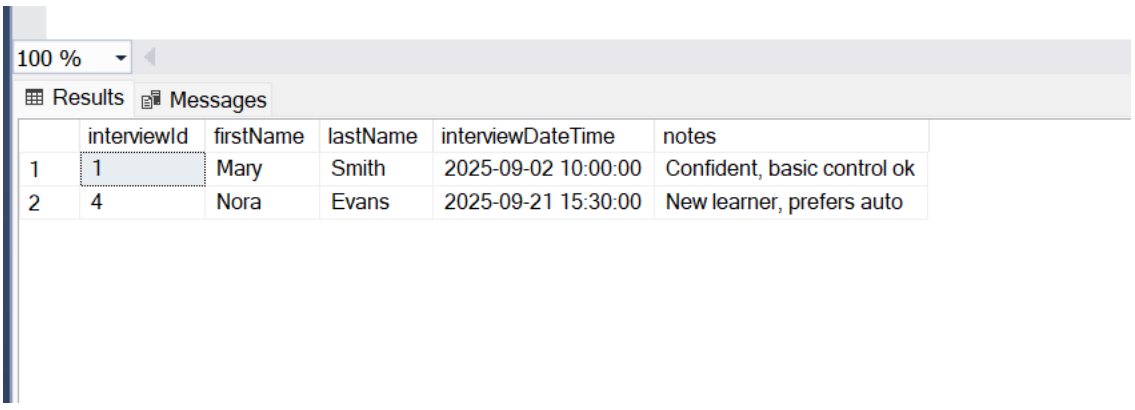
	city	TotalClients
1	Glasgow	4
2	Dundee	1
3	Aberdeen	0

```
-- Query 6: Timetable of appointments for a given Instructor next week (example: instructorId = 4)

SELECT
   l.lessonId,
   l.lessonDate,
```

```
    l.startTime,
    l.endTime,
    c.firstName + ' ' + c.lastName AS ClientName,
    c.phone AS ClientPhone,
    car.regNumber AS CarRegistration,
    a.numberOfLessons,
    a.lessonsRemaining
FROM dbo.Lesson l
JOIN dbo.Client c ON l.clientId = c.clientId
JOIN dbo.Car car ON l.carId = car.carId
LEFT JOIN dbo.Appointment a ON l.appointmentId = a.appointmentId
WHERE l.instructorId = 4
  AND l.lessonDate BETWEEN '2025-11-12' AND '2025-11-15'
ORDER BY l.lessonDate, l.startTime;
```

	lessonId	lessonDate	startTime	endTime	ClientName	ClientPhone	CarRegistration	numberOfLessons	lessonsRemaining
1	1	2025-11-12	09:00:00	10:00:00	Mary Smith	07000-1111	GLA-555	10	8
2	2	2025-11-14	14:00:00	15:00:00	Mary Smith	07000-1111	GLA-555	10	8

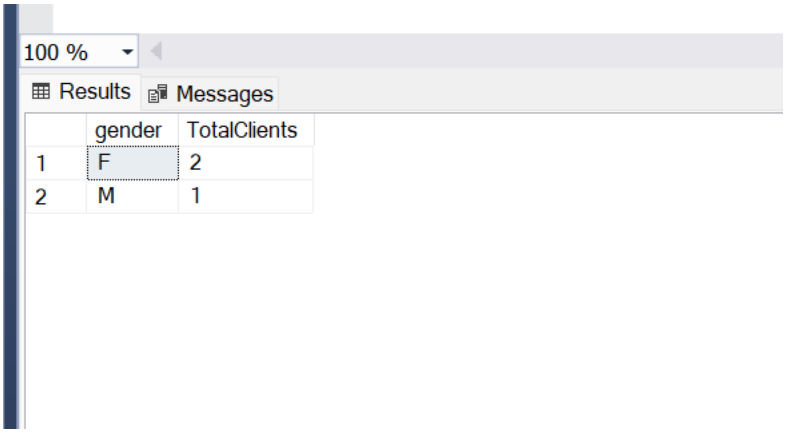
```
-- Query 7: The details of interviews conducted by a given Instructor (example instructorId = 4)

SELECT i.interviewId, c.firstName, c.lastName, i.interviewDateTime, i.notes
FROM dbo.Interview i
JOIN dbo.Client c ON i.clientId = c.clientId
WHERE i.instructorId = 4;
```



	interviewId	firstName	lastName	interviewDateTime	notes
1	1	Mary	Smith	2025-09-02 10:00:00	Confident, basic control ok
2	4	Nora	Evans	2025-09-21 15:30:00	New learner, prefers auto

-- Query 8: The total number of female and male clients (past and present) in the Glasgow, Bearsden office

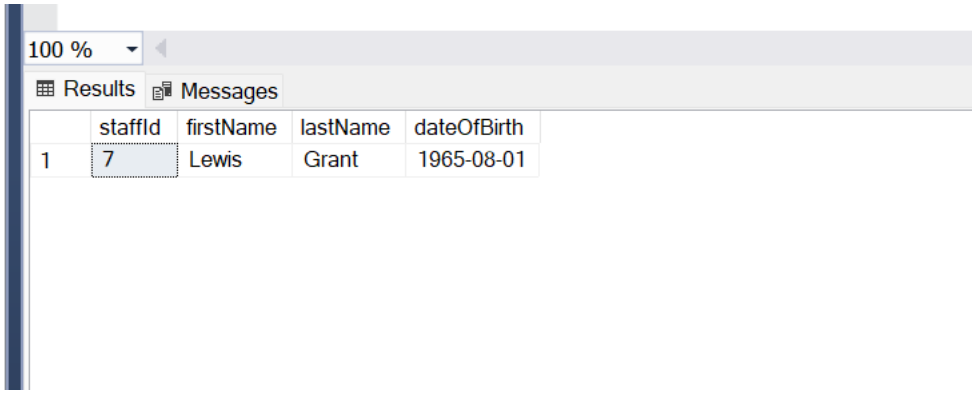
```
SELECT
   c.gender,
   COUNT(c.clientId) AS TotalClients
FROM Client c
INNER JOIN Office o ON c.officeId = o.officeId
WHERE o.officeName = 'Bearsden Office'
GROUP BY c.gender;
```



	gender	TotalClients
1	F	2
2	M	1

-- Query 9: The numbers and name of staff who are Instructors and over 55 years old

```
SELECT staffId, firstName, lastName, dateOfBirth
FROM dbo.Staff
WHERE position LIKE '%Instructor%'
  AND DATEDIFF (YEAR, dateOfBirth, GETDATE ()) > 55;
```

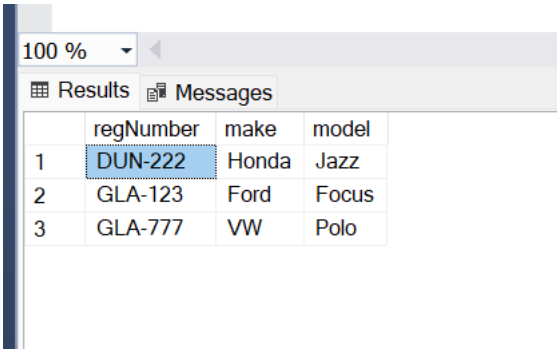


	staffId	firstName	lastName	dateOfBirth
1	7	Lewis	Grant	1965-08-01

-- Query 10: The registration number of cars that have had no faults found

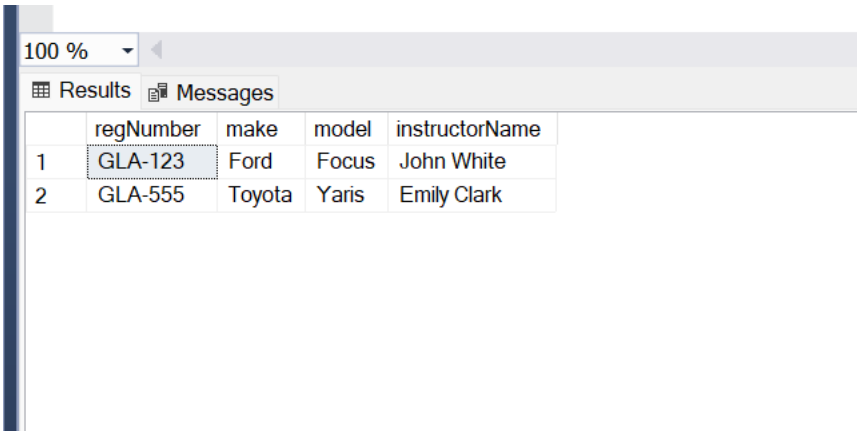
```
SELECT DISTINCT
```

```
    c.regNumber,
    c.make,
    c.model
FROM dbo.Car c
WHERE c.carId NOT IN (
    SELECT DISTINCT carId
    FROM dbo.Inspection
    WHERE faultFound = 1
)
ORDER BY c.regNumber;
```



	regNumber	make	model
1	DUN-222	Honda	Jazz
2	GLA-123	Ford	Focus
3	GLA-777	VW	Polo

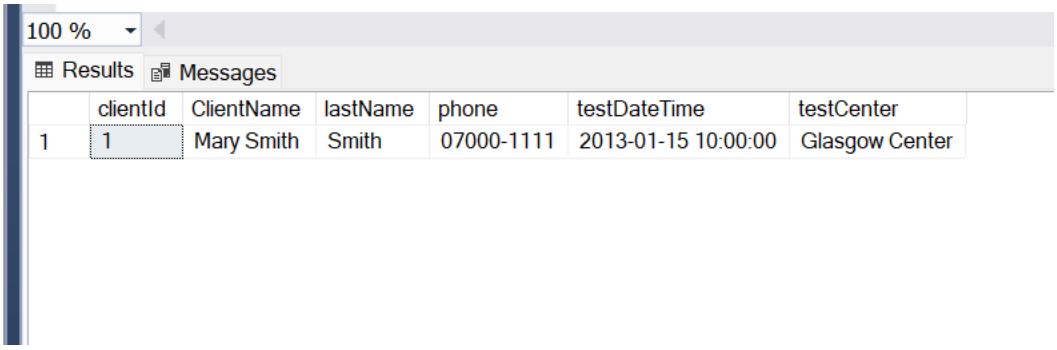
```
-- Query 11: The registration number of the cars used by Instructors at the Glasgow, Bearsden office

SELECT DISTINCT
    car.regNumber,
    car.make,
    car.model,
    s.firstName + ' ' + s.lastName AS instructorName
FROM dbo.Car car
INNER JOIN dbo.Staff s ON car.instructorId = s.staffId
INNER JOIN dbo.Office o ON car.officeId = o.officeId
WHERE o.officeName = 'Bearsden Office'
  AND s.position IN ('Instructor', 'Senior Instructor')
ORDER BY car.regNumber;
```



	regNumber	make	model	instructorName
1	GLA-123	Ford	Focus	John White
2	GLA-555	Toyota	Yaris	Emily Clark

-- Query 12: The names of clients who passed the driving test in January 2013

```
SELECT DISTINCT
   c.clientId,
   c.firstName + ' ' + c.lastName AS ClientName,
   c.lastName,
   c.phone,
   dt.testDateTime,
   dt.testCenter
FROM dbo.Client c
INNER JOIN dbo.DrivingTest dt ON c.clientId = dt.clientId
WHERE dt.drivingTestPassed = 1
  AND dt.writtenTestPassed = 1
  AND YEAR (dt.testDateTime) = 2013
  AND MONTH (dt.testDateTime) = 1
ORDER BY dt.testDateTime, c.lastName;
```

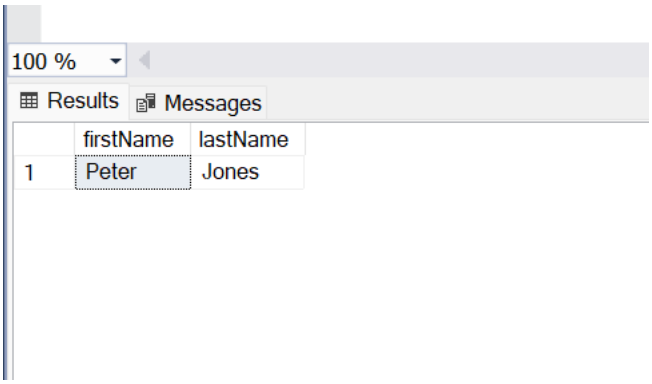


	clientId	ClientName	lastName	phone	testDateTime	testCenter
1	1	Mary Smith	Smith	07000-1111	2013-01-15 10:00:00	Glasgow Center

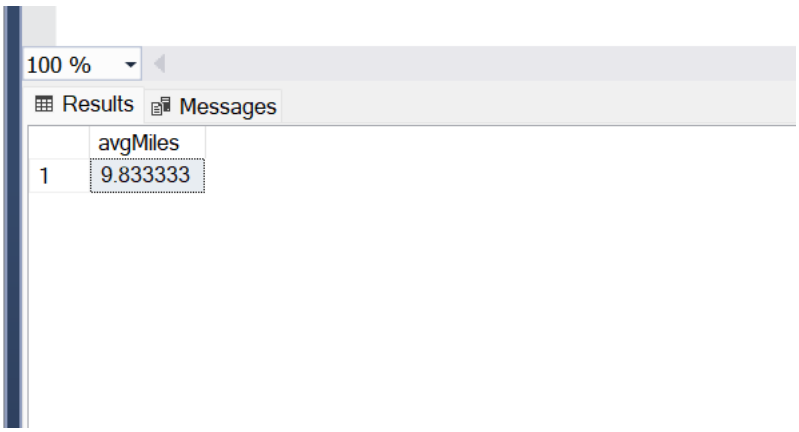
-- Query 13: The names of clients who have sat the driving test more than three times and have still not passed

```
SELECT c.firstName, c.lastName
```

```
FROM dbo.Client c
JOIN dbo.DrivingTest dt ON c.clientId = dt.clientId
WHERE dt.drivingTestPassed = 0
GROUP BY c.clientId, c.firstName, c.lastName
HAVING COUNT (dt.testId) > 3;
```

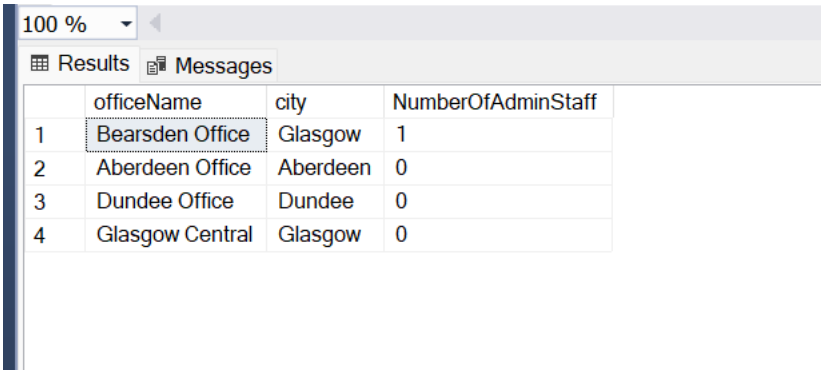



```
-- Query 14: The average number of miles driven during a one-hour lesson

SELECT AVG (milesDriven) AS avgMiles
FROM dbo.Lesson
WHERE DATEDIFF (MINUTE, startTime, endTime) = 60;
```




```
-- Query 15: The number of administrative staff located at each office

SELECT
   o.officeName,
   o.city,
   COUNT(s.staffId) AS NumberOfAdminStaff
FROM Office o
LEFT JOIN Staff s ON o.officeId = s.officeId
   AND s.position = 'Admin'
GROUP BY o.officeId, o.officeName, o.city
```

ORDER BY NumberOfAdminStaff DESC, o.officeName;


100 %

Results Messages

	officeName	city	NumberOfAdminStaff
1	Bearsden Office	Glasgow	1
2	Aberdeen Office	Aberdeen	0
3	Dundee Office	Dundee	0
4	Glasgow Central	Glasgow	0

A. PROJECT PART 2

This part focuses on advanced database features that extend functionality, enforce rules, and automate tasks

- Views: Virtual tables from queries. Simplify access and hide complexity.
- UDFs: Custom functions returning a value or table. Reuse logic in queries.
- Stored Procedures: Precompiled SQL programs with parameters. Automate tasks and enforce business logic.
- Triggers: Automatic actions on INSERT, UPDATE, DELETE. Maintain rules or audit changes.
- Cursors: Process query results row by row. Handle tasks not suited for set-based queries.

1. STORED PROCEDURES

A Stored Procedure (SP) is a program stored in the database. It can include SQL queries + logic.

Characteristics of Stored Procedures

- Can accept parameters
- Can include IF, WHILE, CASE, variables, transactions, loops
- Can insert, update, delete, or select data
- code reuse; don't rewrite SQL once saved
- A stored procedure is compiled once and then saved and made faster
- can hide the underlying tables that protect sensitive data: use EXEC instead of selecting attributes
- makes business logic in one place: avoid duplication and make consistent (no need of manual update, updating is automatic)

```
--1) Lessons for an instructor (all lessons)

CREATE OR ALTER PROCEDURE GetLessonsByInstructorSafe
@InstructorId INT
```

```
AS
BEGIN
  BEGIN TRY
    SELECT * FROM dbo.Lesson WHERE instructorId = @InstructorId;
  END TRY
  BEGIN CATCH
    PRINT 'Error occurred: ' + ERROR_MESSAGE ();
  END CATCH
END;
```

```
EXEC GetLessonsByInstructor 4;
```

100 %

Results | Messages

	lessonId	clientId	instructorId	carId	appointmentId	lessonDate	startTime	endTime	milesDriven	progressNote	lessonType
1	1	1	4	2	1	2025-11-12	09:00:00	10:00:00	12.50	Roundabouts	block
2	2	1	4	2	1	2025-11-14	14:00:00	15:00:00	10.00	Hill start	block
3	6	5	4	2	1	2025-10-21	08:00:00	09:00:00	7.50	Mirrors	individual

--2) Lessons for an instructor in a 7-day period starting from a given date

```
CREATE OR ALTER PROCEDURE dbo.GetLessonsByInstructorDateRange
  @InstructorId INT,
  @StartDate DATE
AS
BEGIN
  SELECT *
  FROM dbo.Lesson
  WHERE instructorId = @InstructorId
    AND lessonDate BETWEEN @StartDate AND DATEADD(DAY, 7, @StartDate);
END;
```

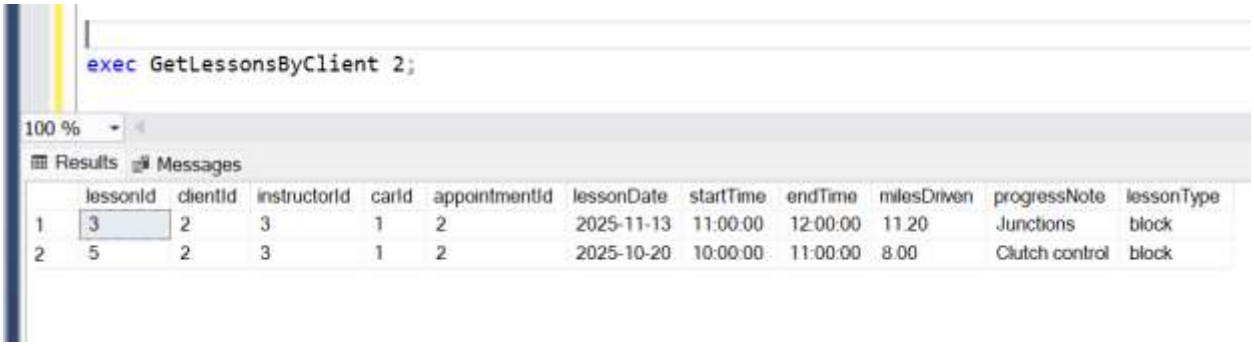
100 %

Results | Messages

	lessonId	clientId	instructorId	carId	appointmentId	lessonDate	startTime	endTime	milesDriven	progressNote	lessonType
1	1	1	4	2	1	2025-11-12	09:00:00	10:00:00	12.50	Roundabouts	block
2	2	1	4	2	1	2025-11-14	14:00:00	15:00:00	10.00	Hill start	block

--3 a) Lessons for a client (all lessons)

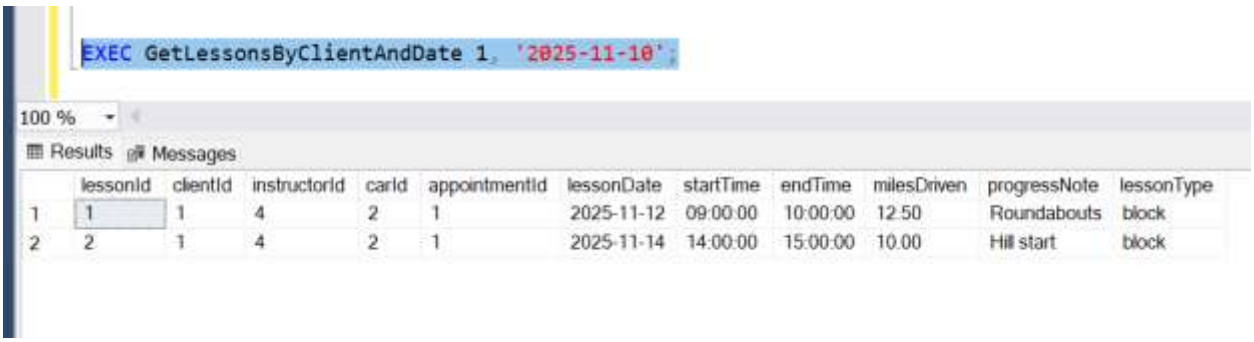
```
CREATE OR ALTER PROCEDURE GetLessonsByClient
  @ClientId INT
AS
BEGIN
  SELECT *
  FROM dbo.Lesson
  WHERE clientId = @ClientId;
END;
```



	lessonId	clientId	instructorId	carId	appointmentId	lessonDate	startTime	endTime	milesDriven	progressNote	lessonType
1	3	2	3	1	2	2025-11-13	11:00:00	12:00:00	11.20	Junctions	block
2	5	2	3	1	2	2025-10-20	10:00:00	11:00:00	8.00	Clutch control	block

--3b) Lessons for client within 7-day window

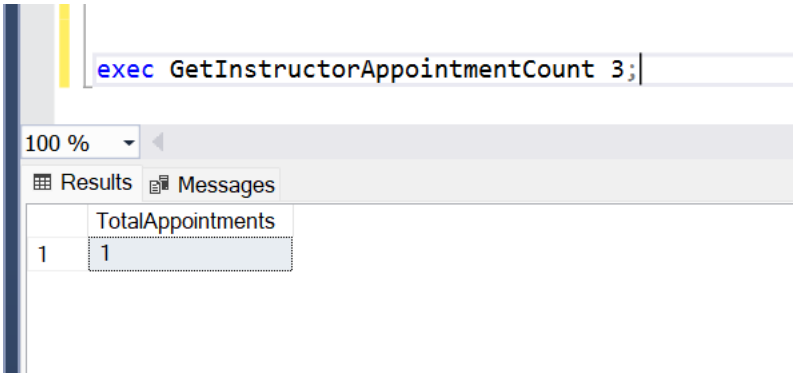
```
CREATE OR ALTER PROCEDURE GetLessonsByClientAndDate
  @ClientId INT,
  @StartDate DATE
AS
BEGIN
  SELECT *
  FROM dbo.Lesson
  WHERE clientId = @ClientId
   AND lessonDate BETWEEN @StartDate AND DATEADD (DAY, 7, @StartDate);
END;
```



	lessonId	clientId	instructorId	carId	appointmentId	lessonDate	startTime	endTime	milesDriven	progressNote	lessonType
1	1	1	4	2	1	2025-11-12	09:00:00	10:00:00	12.50	Roundabouts	block
2	2	1	4	2	1	2025-11-14	14:00:00	15:00:00	10.00	Hill start	block

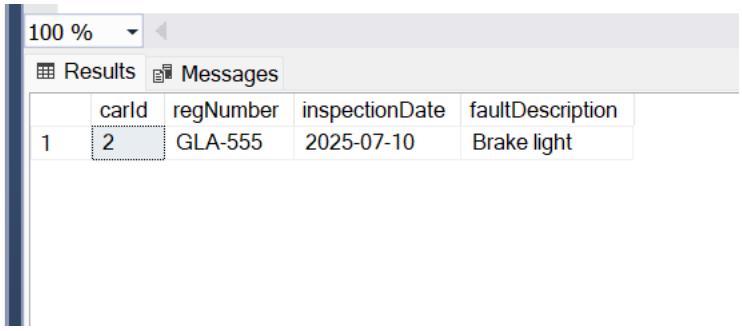
4a) Get total appointments for an instructor

```
CREATE OR ALTER PROCEDURE GetInstructorAppointmentCount
  @InstructorId INT
AS
BEGIN
  SELECT COUNT (*) AS TotalAppointments
  FROM dbo.Appointment
  WHERE instructorId = @InstructorId;
END;
```



4b) Get cars with faults

```
CREATE OR ALTER PROCEDURE GetCarsWithFaults
AS
BEGIN
  SELECT c.carId, c.regNumber, i.inspectionDate, i.faultDescription
  FROM dbo.Car c
  JOIN dbo.Inspection i ON c.carId = i.carId
  WHERE i.faultFound = 1;
END;
```



2. VIEW

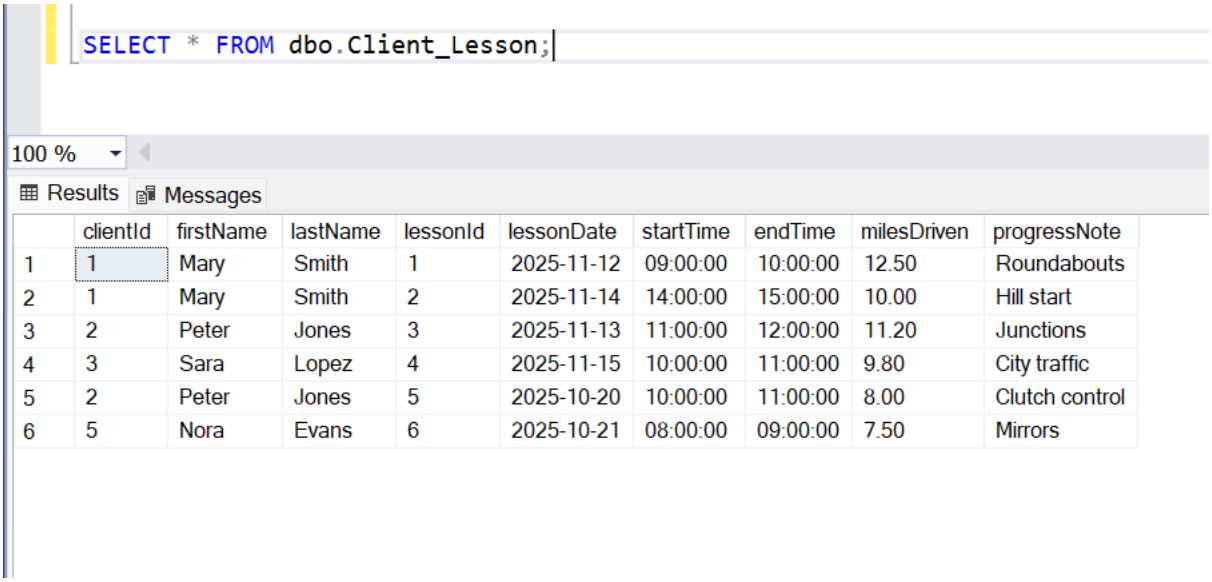
A View is a virtual table based on a SELECT query. It does not store physical data; it only displays data from existing tables.

Characteristics of Views

- Acts like a saved SELECT query
- Can be queried like a table (SELECT * FROM viewName)
- Cannot accept parameters
- Used mainly for simplifying complex joins
- Good for security (hide sensitive columns)
- Automatically updates when underlying tables change

--5) View: Client_Lesson

```
CREATE OR ALTER VIEW dbo.Client_Lesson
AS
SELECT c.clientId, c.firstName, c.lastName,
    l.lessonId, l.lessonDate, l.startTime, l.endTime, l.milesDriven, l.progressNote
FROM dbo.Client c
INNER JOIN dbo.Lesson l ON c.clientId = l.clientId;
```



	clientId	firstName	lastName	lessonId	lessonDate	startTime	endTime	milesDriven	progressNote
1	1	Mary	Smith	1	2025-11-12	09:00:00	10:00:00	12.50	Roundabouts
2	1	Mary	Smith	2	2025-11-14	14:00:00	15:00:00	10.00	Hill start
3	2	Peter	Jones	3	2025-11-13	11:00:00	12:00:00	11.20	Junctions
4	3	Sara	Lopez	4	2025-11-15	10:00:00	11:00:00	9.80	City traffic
5	2	Peter	Jones	5	2025-10-20	10:00:00	11:00:00	8.00	Clutch control
6	5	Nora	Evans	6	2025-10-21	08:00:00	09:00:00	7.50	Mirrors

-- 6) View: Lesson_Info (Extends Client_Lesson with staff info)

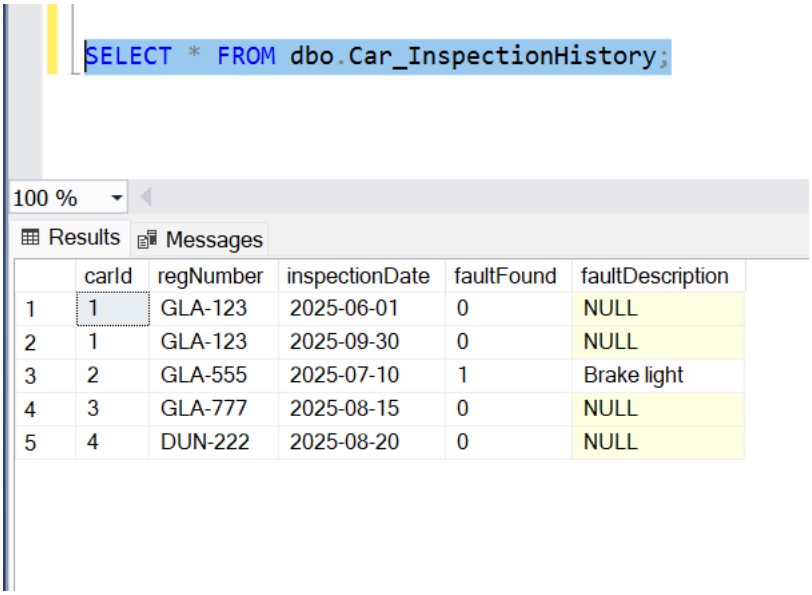
```
CREATE OR ALTER VIEW Lesson_Info AS
```

SELECT cl.*, s.staffId, s.firstName AS InstructorFirstName, s.lastName AS InstructorLastName
FROM Client_Lesson cl
JOIN dbo.Lesson l ON cl.lessonId = l.lessonId
JOIN dbo.Staff s ON l.instructorId = s.staffId;

	clientId	firstName	lastName	lessonId	lessonDate	startTime	endTime	milesDriven	progressNote	staffId	InstructorFirstName	InstructorLastName
1	1	Mary	Smith	1	2025-11-12	09:00:00	10:00:00	12.50	Roundabouts	4	Emily	Clark
2	1	Mary	Smith	2	2025-11-14	14:00:00	15:00:00	10.00	Hill start	4	Emily	Clark
3	2	Peter	Jones	3	2025-11-13	11:00:00	12:00:00	11.20	Junctions	3	John	White
4	3	Sara	Lopez	4	2025-11-15	10:00:00	11:00:00	9.80	City traffic	7	Lewis	Grant
5	2	Peter	Jones	5	2025-10-20	10:00:00	11:00:00	8.00	Clutch control	3	John	White
6	5	Nora	Evans	6	2025-10-21	08:00:00	09:00:00	7.50	Mirrors	4	Emily	Clark

7a) View: Car_InspectionHistory

CREATE OR ALTER VIEW dbo.Car_InspectionHistory
AS
SELECT c.carId, c.regNumber, i.inspectionDate, i.faultFound, i.faultDescription
FROM dbo.Car c
JOIN dbo.Inspection i ON c.carId = i.carId;



	carId	regNumber	inspectionDate	faultFound	faultDescription
1	1	GLA-123	2025-06-01	0	NULL
2	1	GLA-123	2025-09-30	0	NULL
3	2	GLA-555	2025-07-10	1	Brake light
4	3	GLA-777	2025-08-15	0	NULL
5	4	DUN-222	2025-08-20	0	NULL

7b) View: Instructor_Appointments

CREATE OR ALTER VIEW dbo.Instructor_Appointments
AS

```
SELECT s.staffId, s.firstName, s.lastName,
    a.appointmentId, a.purchaseDate, a.numberOfLessons, a.lessonsRemaining
FROM dbo.Staff s
JOIN dbo.Appointment a ON s.staffId = a.instructorId;
```

```
select * from dbo.Instructor_Appointments;
```

100 %

Results | Messages

	staffId	firstName	lastName	appointmentId	purchaseDate	numberOfLessons	lessonsRemaining
1	4	Emily	Clark	1	2025-10-01	10	8
2	3	John	White	2	2025-10-03	5	3
3	7	Lewis	Grant	3	2025-10-05	8	8

3. user defined function

Step: UDF Types in SQL Server

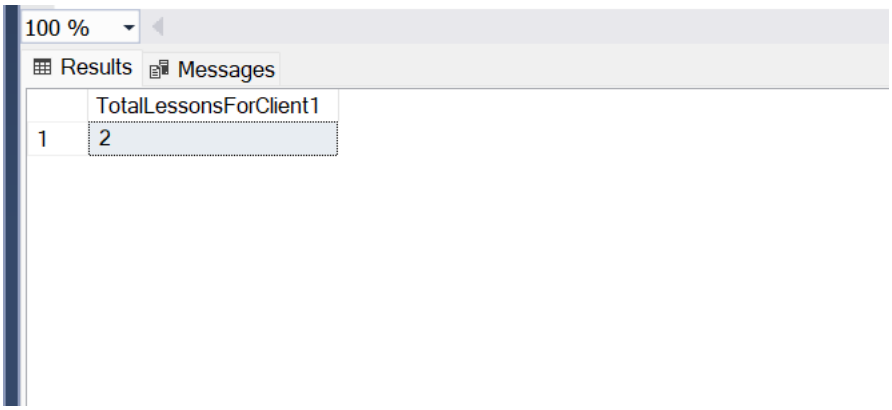
- Scalar UDF – returns a single value.
- Inline Table-Valued Function (TVF) – returns a table (like a view but parameterized).

Character

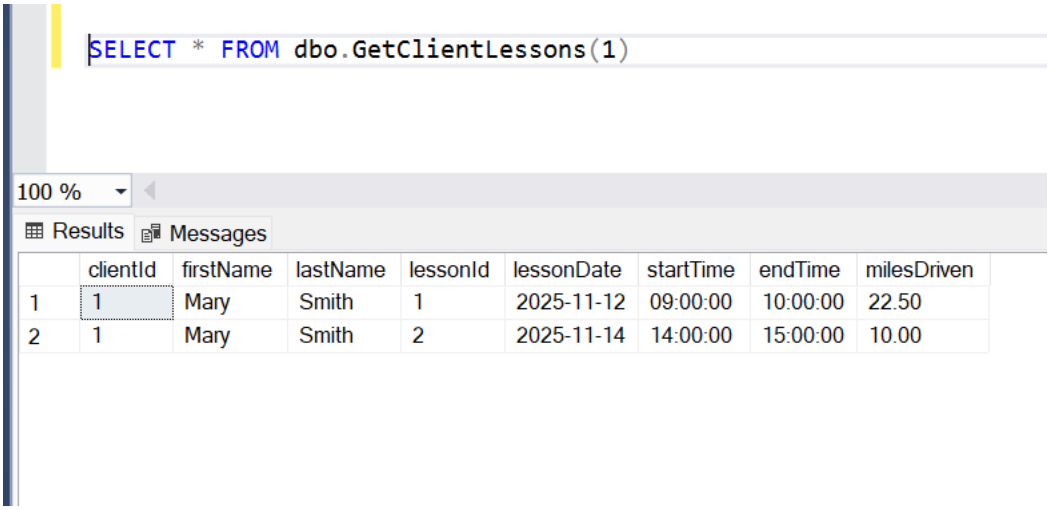
- Can be called from other SQL objects: Stored Procedures, Views, other UDFs can call a UDF
- Reuse logic without rewriting SQL
- Can be used inside SELECT, WHERE, JOIN but Stored procedures cannot do this directly.

```
--8) UDF: Total lessons a client has taken up to today

CREATE OR ALTER FUNCTION dbo.TotalLessonsToDate (@ClientId INT)
RETURNS INT
AS
BEGIN
  DECLARE @Total INT;
  SELECT @Total = COUNT (*)
  FROM dbo.Lesson
  WHERE clientId = @ClientId
   AND lessonDate <= GETDATE ();
```

```
    RETURN @Total;
END;

-- Scalar UDF – returns a single value

SELECT dbo.TotalLessonsToDate (1) AS TotalLessonsForClient1;
```

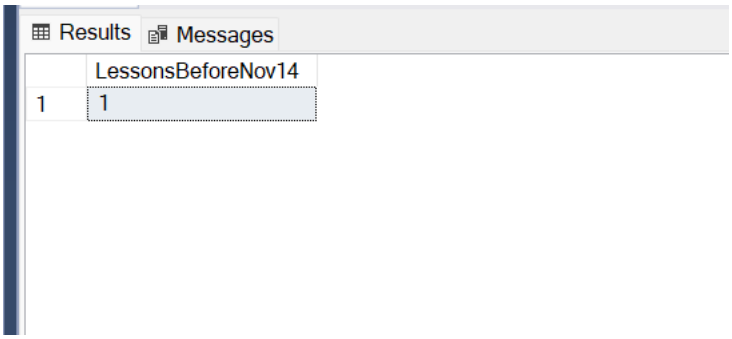


-- **Inline Table-Valued Function**

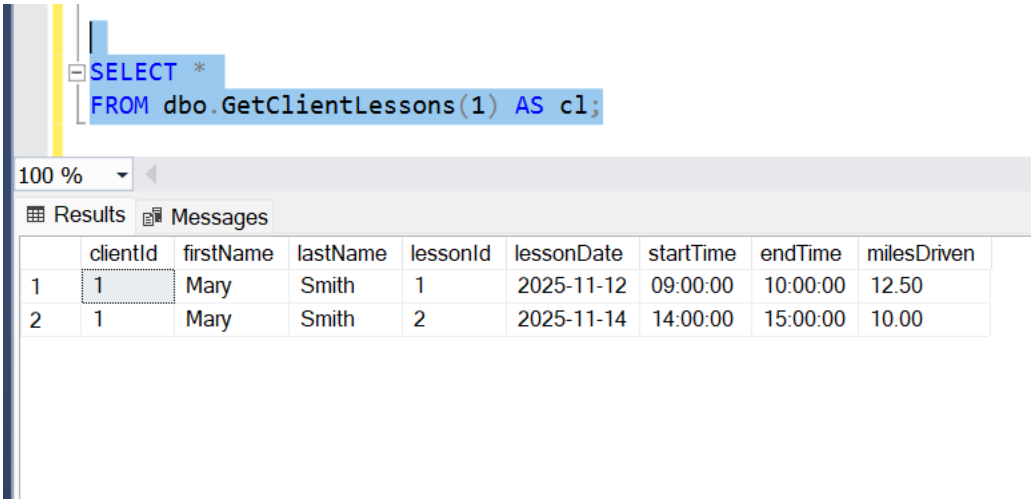



```
-- 9) UDF: Total lessons before a given date

CREATE FUNCTION dbo.TotalLessonsBeforeDate (@ClientId INT, @Date DATE)
RETURNS INT
AS
BEGIN
    DECLARE @Total INT;
    SELECT @Total = COUNT(*)
    FROM dbo.Lesson
```

```
   WHERE clientId = @ClientId
    AND lessonDate < @Date;
   RETURN @Total;
END;

SELECT dbo. TotalLessonsBeforeDate (1, '2025-11-14') AS LessonsBeforeNov14;
```




```
--10) UDF: Return table of Client + Lesson info for a given client

CREATE OR ALTER FUNCTION GetClientLessons (@ClientId INT)
RETURNS TABLE
AS
RETURN
(
   SELECT c.clientId, c.firstName, c.lastName,
        l.lessonId, l.lessonDate, l.startTime, l.endTime, l.milesDriven
   FROM dbo.Client c
   JOIN dbo.Lesson l ON c.clientId = l.clientId
   WHERE c.clientId = @ClientId
);


SELECT *
FROM dbo.GetClientLessons(1) AS cl;
```



	clientId	firstName	lastName	lessonId	lessonDate	startTime	endTime	milesDriven
1	1	Mary	Smith	1	2025-11-12	09:00:00	10:00:00	12.50
2	1	Mary	Smith	2	2025-11-14	14:00:00	15:00:00	10.00

4. Trigger

A trigger is a special type of stored procedure that executes *automatically* in response to INSERT, UPDATE, or DELETE events on a table, while a regular stored procedure must be executed manually using the *EXEC* command.

--11) Trigger – Update instructor's client count

Step 1: Add the new attribute to Staff table

```
ALTER TABLE dbo.Staff
ADD totalClients INT NOT NULL DEFAULT 0;

-- handle a client change from one instructor to another (an UPDATE on instructorId)

CREATE OR ALTER TRIGGER trg_Client_UpdateInstructor
ON dbo.Client
AFTER UPDATE
AS
BEGIN
  IF UPDATE (instructorId)
  BEGIN
    -- Decrement old instructor
    UPDATE s
    SET s.totalClients = s.totalClients - x.cnt
    FROM dbo.Staff s
    JOIN (SELECT instructorId, COUNT (*) AS cnt FROM deleted GROUP BY instructorId) x
     ON s.staffId = x.instructorId;
```

```
    -- Increment of new instructor
    UPDATE s
    SET s.totalClients = s.totalClients + y.cnt
    FROM dbo.Staff s
    JOIN (SELECT instructorId, COUNT (*) AS cnt FROM inserted GROUP BY instructorId) y
      ON s.staffId = y.instructorId;
  END
END;

-- handle new clients being added and clients being removed
```

Step 2: Trigger for INSERT on Client table

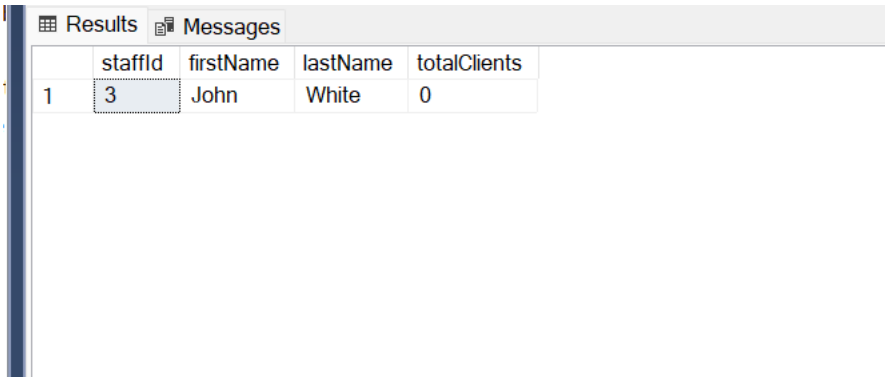
```
CREATE OR ALTER TRIGGER trg_Client_Insert
ON dbo.Client
AFTER INSERT
AS
BEGIN
  -- Increment totalClients for the instructor of each new client
  UPDATE s
  SET s.totalClients = s.totalClients + 1
  FROM dbo.Staff s
  INNER JOIN inserted i ON s.staffId = i.instructorId;
END;
```

Step 3: Trigger for DELETE on Client table

```
CREATE OR ALTER TRIGGER trg_Client_Delete
ON dbo.Client
AFTER DELETE
AS
BEGIN
  -- Decrement totalClients for the instructor of each removed client
  UPDATE s
  SET s.totalClients = s.totalClients - 1
  FROM dbo.Staff s
  INNER JOIN deleted d ON s.staffId = d.instructorId;
END;
```

-- Check initial totalClients for instructor 3

```
SELECT staffId, firstName, lastName, totalClients FROM dbo.Staff WHERE staffId = 3;
```



	staffId	firstName	lastName	totalClients
1	3	John	White	0

-- Insert a new client for instructor 3

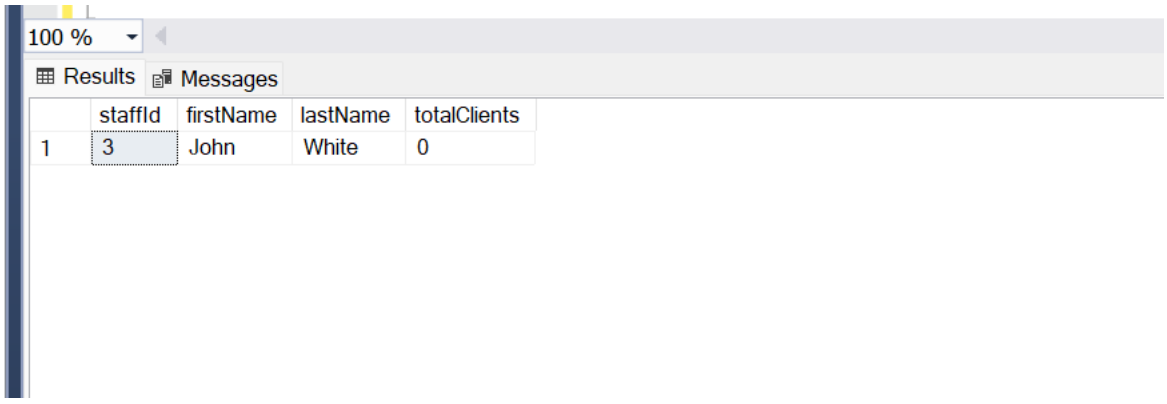
INSERT INTO dbo.Client (officeId, instructorId, firstName, lastName, registrationDate)
VALUES (1, 3, 'Test', 'Client', GETDATE ());

-- Check updated totalClients

SELECT staffId, firstName, lastName, totalClients FROM dbo.Staff WHERE staffId = 3;

	staffId	firstName	lastName	totalClients
1	3	John	White	1

-- Delete the client
DELETE FROM dbo.Client WHERE firstName = 'Test' AND lastName = 'Client';
-- Check the total clients again
SELECT staffId, firstName, lastName, totalClients FROM dbo.Staff WHERE staffId = 3;



	staffId	firstName	lastName	totalClients
1	3	John	White	0

5. Cursor

A cursor is a database object that lets you iterate row by row through the result of a query.

- Normally, SQL works set-based (all rows at once).
- Cursors let you process rows individually, which is useful for row-specific logic.
- Format: declare, open, fetch and close

12) Cursor + IF ... ELSE ... for Lesson mileage

```
DECLARE @LessonId INT, @Miles DECIMAL (6,2),  @ApptId INT, @Fee DECIMAL (10,2);

-- Cursor to iterate through all lessons

DECLARE LessonCursor CURSOR FOR
SELECT lessonId, milesDriven, appointmentId
FROM dbo.Lesson;
OPEN LessonCursor;
FETCH NEXT FROM LessonCursor INTO @LessonId, @Miles, @ApptId;
WHILE @@FETCH_STATUS = 0
BEGIN

   -- Get current totalFee from Appointment

   SELECT @Fee = totalFee
   FROM dbo.Appointment
  WHERE appointmentId = @ApptId;
   -- Apply mileage rules
   IF @Miles > 30
      SET @Fee = @Fee + 10;
   ELSE IF @Miles > 25
      SET @Fee = @Fee + 8;
   ELSE IF @Miles > 20
      SET @Fee = @Fee + 5;

   -- Update Appointment totalFee

   UPDATE dbo.Appointment
   SET totalFee = @Fee
   WHERE appointmentId = @ApptId;
   FETCH NEXT FROM LessonCursor INTO @LessonId, @Miles, @ApptId;
END;
CLOSE LessonCursor;
DEALLOCATE LessonCursor;
```

	appointmentId	clientId	instructorId	carId	numberOfLessons	lessonsRemaining	purchaseDate	discountRate	totalFee
1	1	1	4	2	10	8	2025-10-01	10.00	350.00
2	2	2	3	1	5	3	2025-10-03	0.00	200.00
3	3	3	7	3	8	8	2025-10-05	5.00	280.00

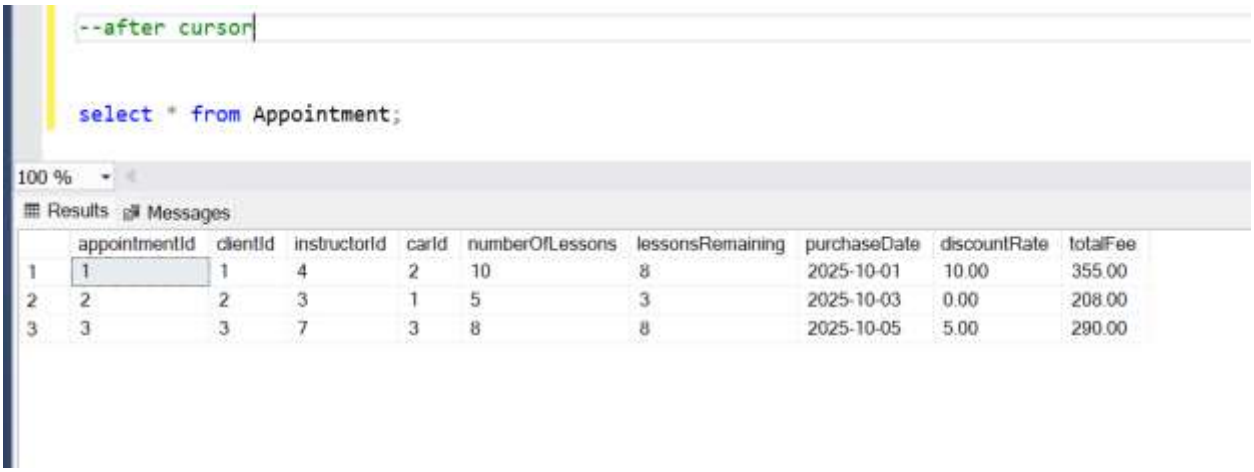
--13) Cursor using CASE

```
DECLARE @LessonId INT, @Miles DECIMAL (6,2), @ApptId INT, @Fee DECIMAL (10,2);

DECLARE LessonCursor CURSOR FOR
SELECT lessonId, milesDriven, appointmentId
FROM dbo.Lesson;
OPEN LessonCursor;
FETCH NEXT FROM LessonCursor INTO @LessonId, @Miles, @ApptId;
WHILE @@FETCH_STATUS = 0
BEGIN

  -- Get current totalFee

  SELECT @Fee = totalFee
  FROM dbo.Appointment
  WHERE appointmentId = @ApptId;

  -- Apply mileage rules using CASE

  SET @Fee = @Fee +
    CASE
      WHEN @Miles > 30 THEN 10
      WHEN @Miles > 25 THEN 8
      WHEN @Miles > 20 THEN 5
      ELSE 0
    END;

  -- Update Appointment

  UPDATE dbo.Appointment
```

```
    SET totalFee = @Fee
    WHERE appointmentId = @ApptId;

    FETCH NEXT FROM LessonCursor INTO @LessonId, @Miles, @ApptId;
END;
CLOSE LessonCursor;
DEALLOCATE LessonCursor;
```

--to show the cursor functionality, our previous seed data is updated here coz all milesDriven first seed data was less than 20

```
UPDATE dbo.Lesson SET milesDriven = 22.5 WHERE lessonId = 1;
UPDATE dbo.Lesson SET milesDriven = 28.0 WHERE lessonId = 3;
UPDATE dbo.Lesson SET milesDriven = 31.5 WHERE lessonId = 4;
```



```
--after cursor

select * from Appointment;
```

100 %

Results | Messages

	appointmentId	clientId	instructorId	carId	numberOfLessons	lessonsRemaining	purchaseDate	discountRate	totalFee
1	1	1	4	2	10	8	2025-10-01	10.00	355.00
2	2	2	3	1	5	3	2025-10-03	0.00	208.00
3	3	3	7	3	8	8	2025-10-05	5.00	290.00

Key takeaway

- Before: totalFee is the original amount when the appointment was created.
- After: totalFee is increased only for appointments whose lessons had miles > 20, 25, or 30.
- While cursors are useful for demonstrating row-by-row logic, in production environments set-based operations are generally preferred for efficiency and scalability.

N.B:

- @variable is a user-defined variable
- @@variable is a system-defined global variable (example Value 0: FETCH was successful, value -1: FETCH failed (no more rows) and value -2: Row is missing).

Conclusion

The EasyDrive School of Motoring database project demonstrates the design and implementation of a complete relational database system. Part 1 covers ER diagrams, logical database design, normalization, and SQL implementation, effectively modeling real-world driving school operations. Part 2 highlights advanced database concepts such as stored procedures, views, user-defined functions, triggers, and cursors, enabling automated, secure, and efficient data management. Overall, this project reinforces practical database skills and illustrates how relational databases streamline complex business operations.