

The EasyDrive School of Motoring

Case Study

The EasyDrive School of Motoring was established in Glasgow in 1992. Since then, the school has grown steadily and now has several offices in most of the main cities of Scotland. However, the school is now so large that more and more administrative staff are being employed to cope with the ever-increasing amount of paperwork. Furthermore, the communication and sharing of information between offices, even in the same city, is poor. The Director of the school, Dave MacLeod, feels that too many mistakes are being made and that the success of the school will be short-lived if he does not do something to remedy the situation. He knows that a database could help in part to solve the problem and has approached you and your team to help in creating a database system to support the running of the EasyDrive School of Motoring. The Director has provided the following brief description of how the EasyDrive School of Motoring operates.

B.2.1 Data Requirements

Each office has a Manager (who tends to also be a Senior Instructor), several Senior Instructors, Instructors, and administrative staff. The Manager is responsible for the day-to-day running of the office. Clients must first register at an office, which includes completion of an application form, which records their personal details. Before the first lesson, a client is requested to attend an interview with an Instructor to assess the needs of the client and to ensure that the client holds a valid provisional driving license. A client is free to ask for a particular Instructor or to request that an Instructor be changed at any stage throughout the process of learning to drive. After the interview, the first lesson is booked. A client may request individual lessons or book a block of lessons for a reduced fee. An individual lesson is for one hour, which begins and ends at the office. A lesson is with a particular Instructor in a particular car at a given time. Lessons can start as early as 8:00 a.m. and as late as 8:00 p.m. After each lesson, the Instructor records the progress made by the client and notes the mileage used during the lesson. The school has a pool of cars, which are adapted for the purposes of teaching. Each Instructor is allocated to a particular car. As well as teaching, the Instructors are free to use the cars for personal use. The cars are inspected at regular intervals for

faults. Once ready, a client applies for a driving test date. To obtain a full driving license, the client must pass both the driving and written parts of the test. It is the

responsibility of the Instructor to ensure that the client is best prepared for all aspects of the test. The Instructor is not responsible for testing the client and is not in the car during the test, but should be available to drop off and pick up the client before and after the test at the Testing Center. If a client fails to pass, the Instructor must record the reasons for the failure.

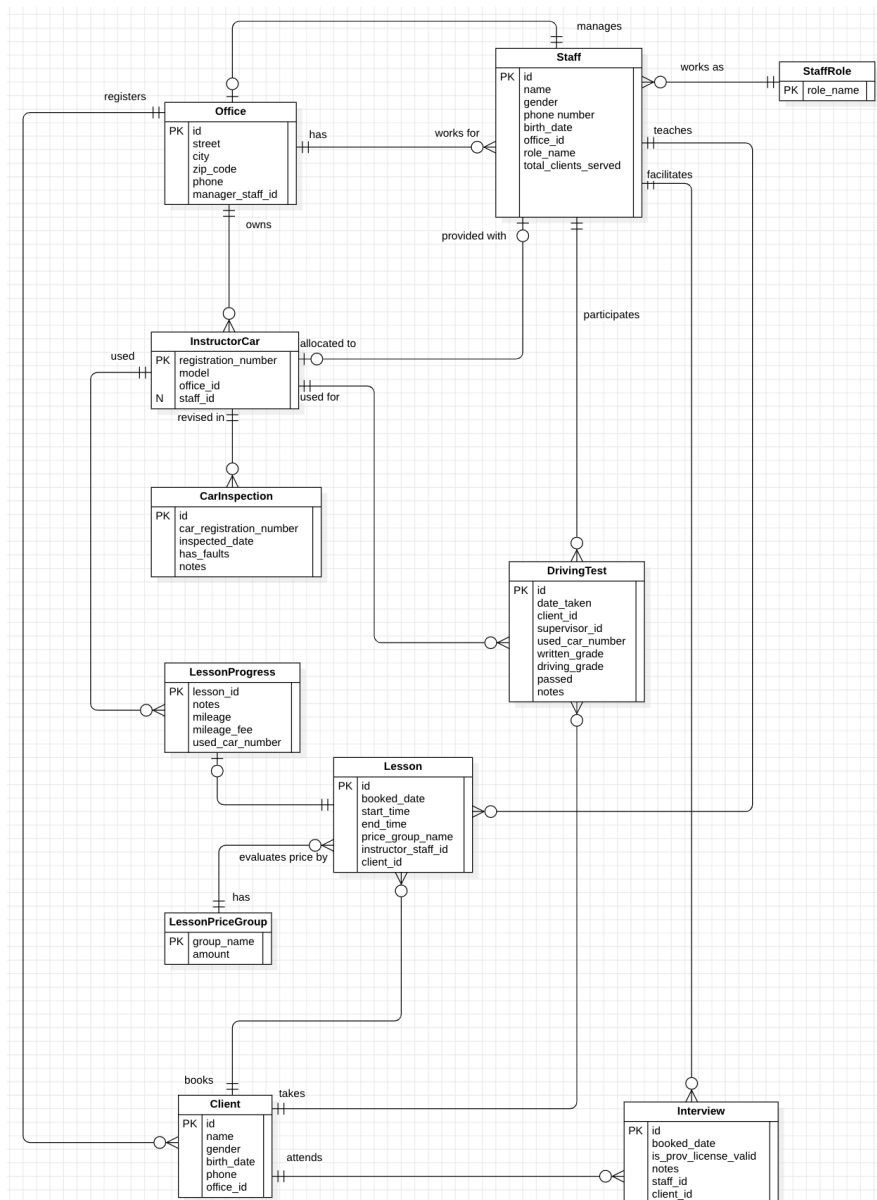
B.2.2 Query Transactions (Sample)

The director has provided some examples of typical queries that the database system for the EasyDrive School of Motoring must support:

- (a) The names and the telephone numbers of the Managers of each office.
- (b) The full address of all offices in Glasgow.
- (c) The names of all female Instructors based in the Glasgow, Bearsden office.
- (d) The total number of staff at each office.
- (e) The total number of clients (past and present) in each city.
- (f) The timetable of appointments for a given Instructor next week.
- (g) The details of interviews conducted by a given Instructor.
- (h) The total number of female and male clients (past and present) in the Glasgow, Bearsden office.
- (i) The numbers and name of staff who are Instructors and over 55 years old.
- (j) The registration number of cars that have had no faults found.

- (k) The registration number of the cars used by Instructors at the Glasgow, Bearsden office.
- (l) The names of clients who passed the driving test in January 2013.
- (m) The names of clients who have sat the driving test more than three times and have still not passed.
- (n) The average number of miles driven during a one-hour lesson,
- (o) The number of administrative staff located at each office.

ER Diagram



Logical DB

Office(id, street, city, zip_code, phone, manager_staff_id)

Primary Key: id

Foreign Key: manager_staff_id references Staff(id)

Alternate Key: manager_staff_id

InstructorCar(registration_number, model, office_id, staff_id)

Primary Key: registration_number

Foreign Key: office_id references Office(id)

Foreign Key: staff_id references Staff(id)

Alternate Key: staff_id

CarInspection(id, registration_number, inspected_date, has_faults, notes)

Primary Key: id

Foreign Key: registration number references InstructorCar(registration_number)

Staff(id, name, gender, phone_number, birth_date, office_id, role_name, total_clients_served)

Primary Key: id

Foreign Key: office_id references Office(id)

StaffRole(role_name)

Primary Key: role_name

Client(id, name, gender, birth_date, phone, office_id)

Primary Key: id

Foreign Key: office_id references Office(id)

Interview(id, booked_date, is_prov_license_valid, notes, staff_id, client_id, book_time)

Primary Key: id

Foreign Key: staff_id references Staff(id)

Foreign Key: client_id references Client(id) d

Lesson(id, booked_date, start_time, end_time, price_group_name, instructor_staff_id, client_id)

Primary Key: id

Foreign Key: instructor_staff_id references Staff(id)

Foreign Key: client_id references Client(id)

Foreign Key: price_group_name references LessonPriceGroup(group_name)

LessonPriceGroup(group_name, amount)

Primary Key: group_name

LessonProgress(lesson_id, notes, mileage, mileage_fee, used_car_number)

Primary Key: lesson_id

Foreign Key: used_car_number references InstructorCar(registration_number)

DrivingTest(id, date_taken, client_id, supervisor_id, used_car_number, written_grade, driving_grade, passed, notes)

Primary Key: id

Foreign Key: client_id references Client(id)

Foreign Key: supervisor_id references Staff(id)

Foreign Key: used_car_number references InstructorCar(registration_number)

Normalization

Resultant Logical DB is at the highest normalizable state. However, there are arguable solutions that are need to be mentioned:

1. From the table InstructorCar:

FFD:

(registration_number) -> model, office_id, staff_id

Partial dependencies:

(staff_id) -> office_id

In initial review, it's probably needed to eliminate this dependency since "office_id" is already available within the Staff table. However, according to the ER diagram, a car may or may not be assigned to a particular staff member due to several cases,

for example, an office has recently been established owning some cars, quantity of which exceeds number of available staff members.

Therefore, if we eliminate “office_id”, the Chasm trap may occur and we will not be able to show, for example, number of cars per office since there is not enough of staff members.

2. The tables Lesson and DrivingTest have a common case. It’s possible to identify following partial dependencies:

(supervisor_id) -> used_car_number

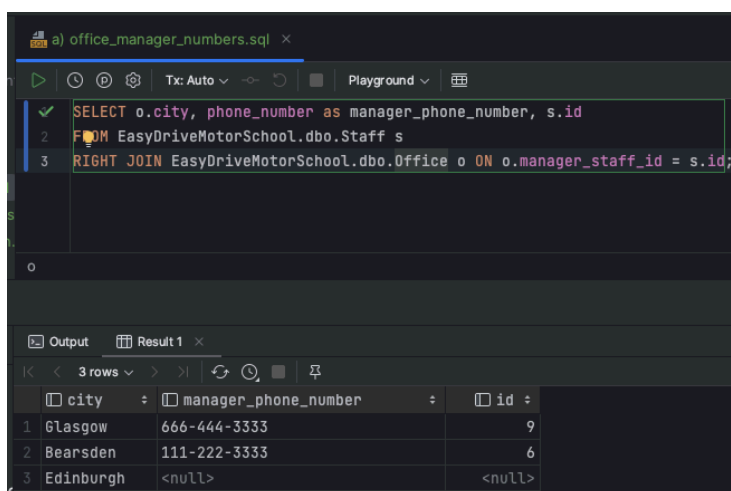
(used_car_number) -> supervisor_id

Obviously, one of the attributes could be eliminated so we avoid partial dependencies. Although, following cases can prove that these partial dependencies are not Full Functional Dependencies:

1. During either a lesson or driving test, instructor of “supervisor_id” may not provide own assigned car because, as an example, an office may have number of instructors greater than number of cars so, for a purpose of a lesson or a driving test, any other available car will be used.
2. A car of an instructor can be restricted of use if any faults are identified after the car inspection process. So in that case, there is an option for an office to provide another available car or, if there are no available cars, to reschedule a lesson/a driving test.

Queries:

- (a) The names and the telephone numbers of the Managers of each office.



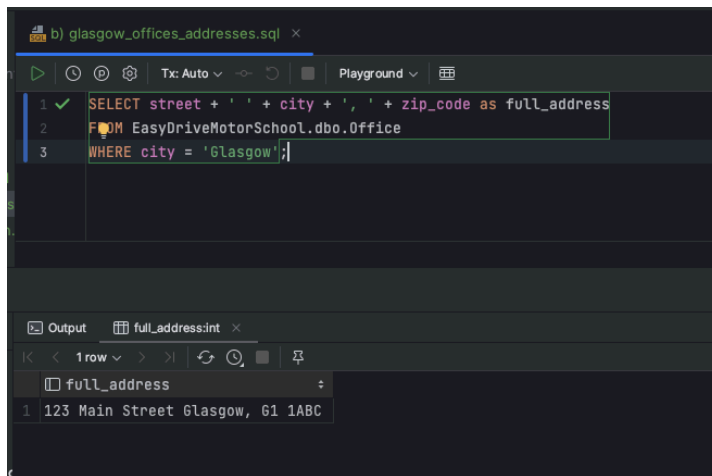
The screenshot shows a SQL query editor with a query titled 'a) office_manager_numbers.sql'. The query is as follows:

```
SELECT o.city, phone_number as manager_phone_number, s.id
FROM EasyDriveMotorSchool.dbo.Staff s
RIGHT JOIN EasyDriveMotorSchool.dbo.Office o ON o.manager_staff_id = s.id;
```

The results are displayed in a table with 3 rows and 3 columns: city, manager_phone_number, and id.

city	manager_phone_number	id
Glasgow	666-444-3333	9
Bearsden	111-222-3333	6
Edinburgh	<null>	<null>

(b) The full address of all offices in Glasgow.



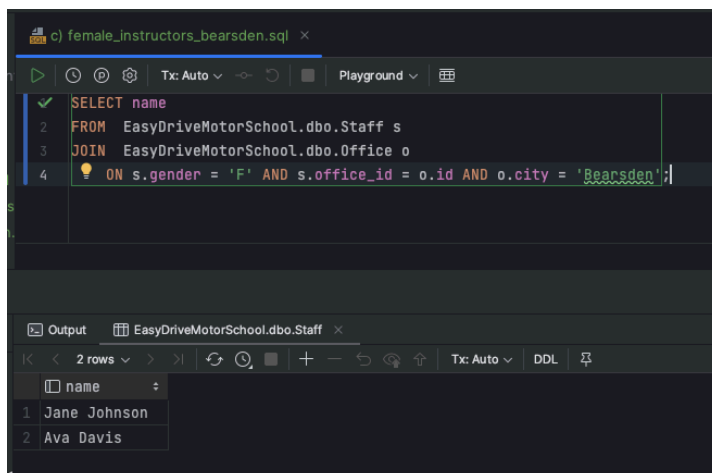
The screenshot shows a SQL playground interface. The top panel contains a query file named 'b) glasgow_offices_addresses.sql'. The query is as follows:

```
1 SELECT street + ' ' + city + ', ' + zip_code as full_address
2 FROM EasyDriveMotorSchool.dbo.Office
3 WHERE city = 'Glasgow';
```

The bottom panel shows the output of the query. It displays a table with one row and one column named 'full_address'. The value in the row is '123 Main Street Glasgow, G1 1ABC'.

full_address
123 Main Street Glasgow, G1 1ABC

(c) The names of all female Instructors based in the Glasgow, Bearsden office.



The screenshot shows a SQL playground interface. The top panel contains a query file named 'c) female_instructors_bearsden.sql'. The query is as follows:

```
1 SELECT name
2 FROM EasyDriveMotorSchool.dbo.Staff s
3 JOIN EasyDriveMotorSchool.dbo.Office o
4 ON s.gender = 'F' AND s.office_id = o.id AND o.city = 'Bearsden';
```

The bottom panel shows the output of the query. It displays a table with two rows and one column named 'name'. The values in the rows are 'Jane Johnson' and 'Ava Davis'.

name
Jane Johnson
Ava Davis

(d) The total number of staff at each office.

The screenshot shows a SQL query in a playground environment. The query is as follows:

```
1 SELECT
2   o.street + ' ' + o.city as office,
3   (
4     SELECT COUNT(*)
5     FROM EasyDriveMotorSchool.dbo.Staff s
6     WHERE s.office_id = o.id
7   ) as staff_number
8 FROM EasyDriveMotorSchool.dbo.Office o;
```

The result set, titled 'Result 1', contains 3 rows:

	office	staff_number
1	123 Main Street Glasgow	8
2	456 Oak Street Bearsden	7
3	55 Never Street Edinburgh	0

(e) The total number of clients (past and present) in each city.

The screenshot shows a SQL query in a playground environment. The query is as follows:

```
1 SELECT
2   city,
3   COUNT(c.office_id) as clients_number
4 FROM EasyDriveMotorSchool.dbo.Office o
5 LEFT JOIN EasyDriveMotorSchool.dbo.Client c ON c.office_id = o.id
6 GROUP BY city;
```

The result set, titled 'Result 1', contains 3 rows:

	city	clients_number
1	Bearsden	6
2	Edinburgh	0
3	Glasgow	7

(f) The timetable of appointments for a given Instructor next week.

The screenshot shows a SQL IDE with a stored procedure named `InstructorAppointmentsNextWeek` and its execution results.

```

1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER PROCEDURE InstructorAppointmentsNextWeek
5     @instructor_id INT,
6     @date_requested DATE
7 AS
8 BEGIN
9     SELECT
10         CONCAT(l.booked_date, ' ', RIGHT(CONVERT(VARCHAR, l.booked_time, 100), 7)) as date_time,
11         c.id as client_id, c.name as client_name, 'Interview' as appointment_type
12     FROM EasyDriveMotorSchool.dbo.Interview i
13     INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = i.client_id
14     WHERE (l.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0) -- Start of next week
15           AND l.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND i.staff_id = @instructor_id -- Start of the week after next
16
17     UNION
18
19     SELECT
20         CONCAT(l.booked_date, ' ', RIGHT(CONVERT(VARCHAR, l.start_time, 100), 7)) as date_time,
21         c2.id as client_id, c2.name as client_name, 'Lesson' as appointment_type
22     FROM EasyDriveMotorSchool.dbo.Lesson l
23     INNER JOIN EasyDriveMotorSchool.dbo.Client c2 on c2.id = l.client_id
24     WHERE (l.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0) -- Start of next week
25           AND l.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND l.instructor_staff_id = @instructor_id -- Start of the week after next
26
27     UNION
28
29     SELECT
30         t.date_taken as date_time,
31         c2.id as client_id, c2.name as client_name, 'Driving Test' as appointment_type
32     FROM EasyDriveMotorSchool.dbo.DrivingTest t
33     INNER JOIN EasyDriveMotorSchool.dbo.Client c2 on c2.id = t.client_id
34     WHERE (t.date_taken >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0) -- Start of next week
35           AND t.date_taken < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND t.supervisor_id = @instructor_id -- Start of the week after next
36
37 END
38 GO;
39
40 EXEC InstructorAppointmentsNextWeek @instructor_id = 1, @date_requested = '2023-11-1';
  
```

The output window shows the following results:

date_time	client_id	client_name	appointment_type
2023-11-08	1	John Doe	Interview
2023-11-08	2	Jane Smith	Interview
2023-11-09	1	John Doe	Lesson
2023-11-11	1	John Doe	Driving Test
2023-11-12	1	John Doe	Driving Test

(g) The details of interviews conducted by a given Instructor.

The screenshot shows a SQL IDE with a stored procedure named `InstructorInterviewDetails` and its execution results.

```

1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER PROCEDURE InstructorInterviewDetails
5     @instructor_id INT
6 AS
7 BEGIN
8     SELECT
9         CONCAT(i.booked_date, ' ', RIGHT(CONVERT(VARCHAR, i.booked_time, 100), 7)) as date_time,
10        c.id as client_id, c.name as client_name, i.notes, i.is_prov_license_valid
11    FROM EasyDriveMotorSchool.dbo.Interview i
12    INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = i.client_id
13    WHERE i.staff_id = @instructor_id
14
15 END
16 GO;
17
18 EXEC InstructorInterviewDetails @instructor_id = 1;
  
```

The output window shows the following results:

date_time	client_id	client_name	notes	is_prov_license_valid
2023-11-08 8:33AM	1	John Doe	OK	true
2023-11-08 9:34AM	2	Jane Smith	OK	true

(h) The total number of female and male clients (past and present) in the Glasgow, Bearsden office.

```
h) female_and_male_clients_number_glasgow.sql x
1 SELECT city,
2     COUNT(CASE WHEN c.gender = 'M' THEN 1 END) AS count_male,
3     COUNT(CASE WHEN c.gender = 'F' THEN 1 END) AS count_female
4 FROM EasyDriveMotorSchool.dbo.Office o
5 INNER JOIN EasyDriveMotorSchool.dbo.Client c on o.id = c.office_id
6 GROUP BY city;
```

Output

date_time	client_id	client_name	notes	is_prov_license_valid
2023-11-08 8:33AM	1	John Doe	OK	• true
2023-11-08 9:34AM	2	Jane Smith	OK	• true

(i) The numbers and name of staff who are Instructors and over 55 years old.

```
i) number_and_name_staff_older_55.sql x
1 ✓ SELECT s.name, s.phone_number, DATEDIFF(YEAR, s.birth_date, GETDATE()) as age
2 FROM EasyDriveMotorSchool.dbo.Staff s
3 WHERE (role_name = 'Instructor' OR role_name = 'Senior Instructor') AND DATEDIFF(YEAR, s.birth_date, GETDATE()) >= 55;
4
```

Output

name	phone_number	age
Alice Brown	555-123-4567	58
Michael White	111-222-3333	63
Olivia Lee	666-444-3333	55
Jordan Taylor	555-555-5555	60
Ava Davis	555-123-4567	56

(j) The registration number of cars that have had no faults found.

The screenshot shows a SQL IDE with a query titled 'reg_number_cars_with_no_faults.sql'. The query is as follows:

```
1 SELECT
2     c.registration_number
3 FROM EasyDriveMotorSchool.dbo.InstructorCar c
4 INNER JOIN EasyDriveMotorSchool.dbo.CarInspection ci on c.registration_number = ci.registration_number
5 GROUP BY c.registration_number
6 HAVING COUNT(CASE WHEN ci.has_faults = 1 THEN 1 END) = 0
```

The output window shows the results of the query, displaying the registration numbers of three cars:

registration_number
1 CAR002
2 CAR003
3 CAR004

(k) The registration number of the cars used by Instructors at the Glasgow, Bearsden office.

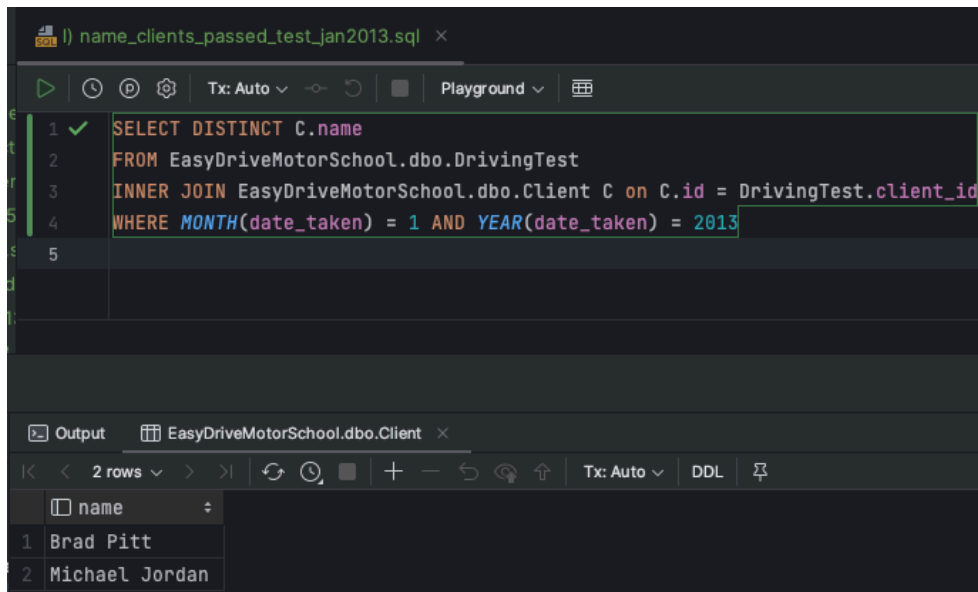
The screenshot shows a SQL IDE with a query titled 'cars_used_by_instructors_bearsden.sql'. The query is as follows:

```
1 SELECT registration_number
2 FROM EasyDriveMotorSchool.dbo.InstructorCar ic
3 INNER JOIN EasyDriveMotorSchool.dbo.Staff s
4     on s.id = ic.staff_id AND (s.role_name = 'Instructor' OR s.role_name = 'Senior Instructor')
5 INNER JOIN EasyDriveMotorSchool.dbo.Office o on o.id = ic.office_id AND o.city = 'Bearsden'
```

The output window shows the results of the query, displaying the registration numbers of five cars:

registration_number
1 CAR001
2 CAR002
3 CAR003
4 CAR004
5 CAR005

(l) The names of clients who passed the driving test in January 2013.



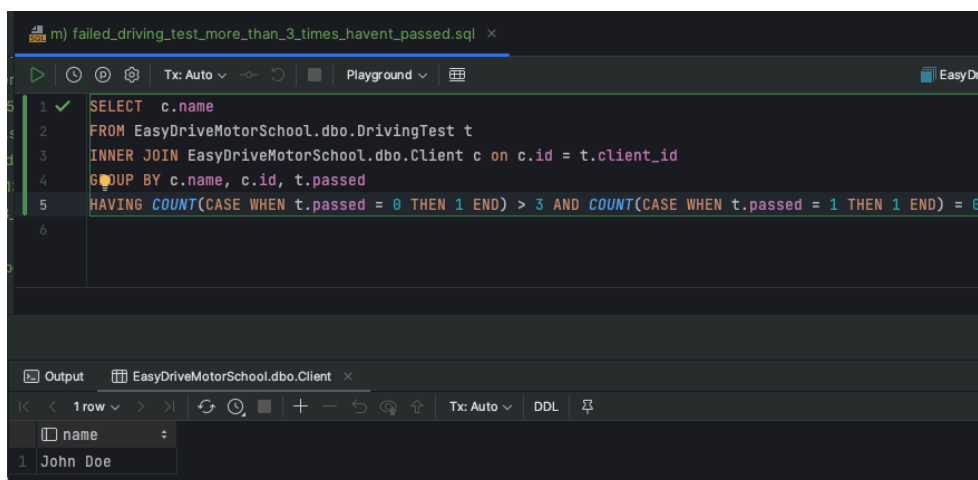
The screenshot shows a SQL IDE with a query editor and an output window. The query editor contains the following SQL code:

```
1 SELECT DISTINCT C.name
2 FROM EasyDriveMotorSchool.dbo.DrivingTest
3 INNER JOIN EasyDriveMotorSchool.dbo.Client C on C.id = DrivingTest.client_id
4 WHERE MONTH(date_taken) = 1 AND YEAR(date_taken) = 2013
5
```

The output window shows the results of the query, which are two rows:

name
1 Brad Pitt
2 Michael Jordan

(m) The names of clients who have sat the driving test more than three times and have still not passed.



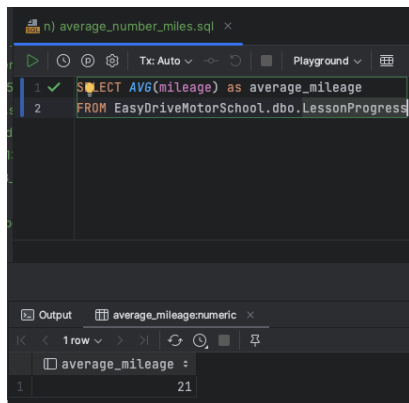
The screenshot shows a SQL IDE with a query editor and an output window. The query editor contains the following SQL code:

```
1 SELECT c.name
2 FROM EasyDriveMotorSchool.dbo.DrivingTest t
3 INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = t.client_id
4 GROUP BY c.name, c.id, t.passed
5 HAVING COUNT(CASE WHEN t.passed = 0 THEN 1 END) > 3 AND COUNT(CASE WHEN t.passed = 1 THEN 1 END) = 0
6
```

The output window shows the results of the query, which is one row:

name
1 John Doe

(n) The average number of miles driven during a one-hour lesson,

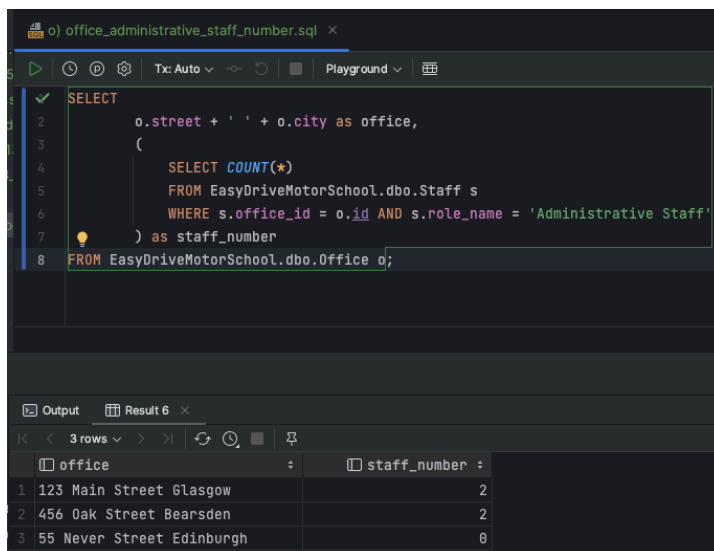


```
n) average_number_miles.sql
1 SELECT AVG(mileage) as average_mileage
2 FROM EasyDriveMotorSchool.dbo.LessonProgress
```

Output: average_mileage:numeric

average_mileage
21

(o) The number of administrative staff located at each office.



```
o) office_administrative_staff_number.sql
1 SELECT
2     o.street + ' ' + o.city as office,
3     (
4         SELECT COUNT(*)
5         FROM EasyDriveMotorSchool.dbo.Staff s
6         WHERE s.office_id = o.id AND s.role_name = 'Administrative Staff'
7     ) as staff_number
8 FROM EasyDriveMotorSchool.dbo.Office o;
```

Output: Result 6

office	staff_number
123 Main Street Glasgow	2
456 Oak Street Bearsden	2
55 Never Street Edinburgh	0

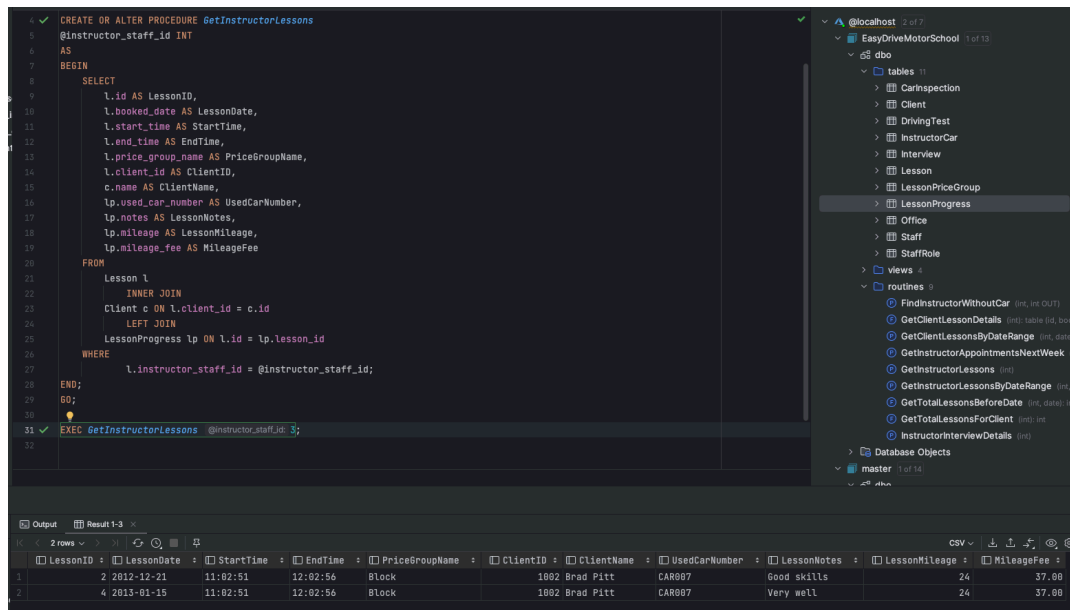
Part 2:

PART 2

STORED PROCEDURES

- 1) Write a stored procedure that takes in one argument, the staff number of an instructor. The procedure outputs all details of all the lessons for that instructor.
- 2) Write a stored procedure that takes in two arguments, a staff number and a date. The procedure shows details of all lessons for that staff instructor, starting at the date of the argument, and ending seven days later.
- 3) Do the same as questions 1 and 2 above, but for a client number instead of a staff number.
- 4) Create some stored procedures yourself which do something you would like to see being done.

1)



```

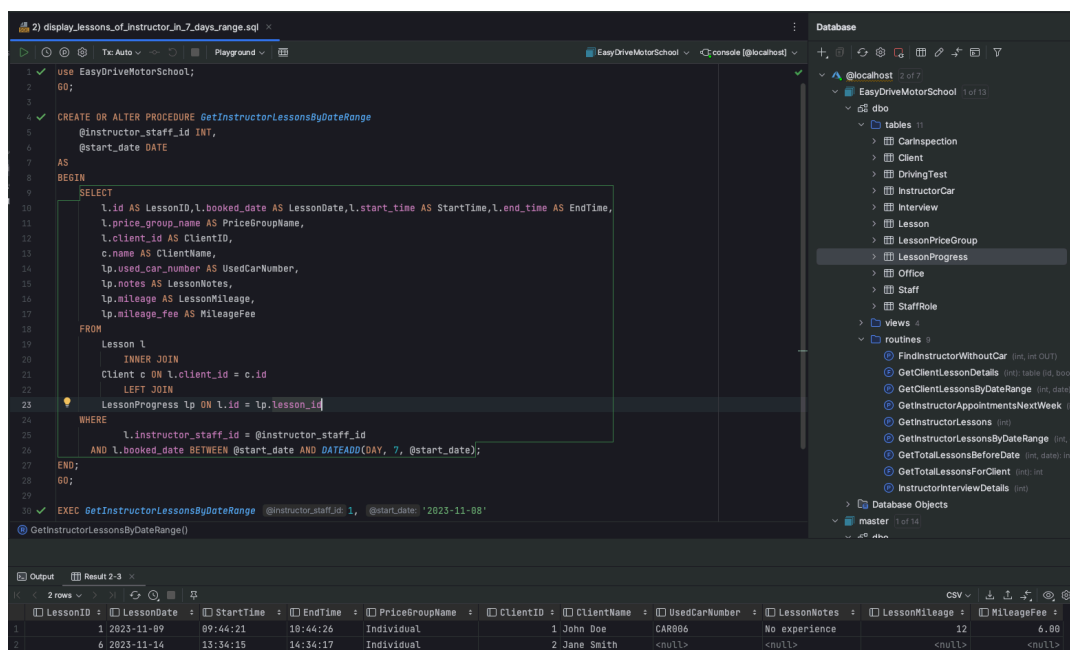
CREATE OR ALTER PROCEDURE GetInstructorLessons
    @Instructor_staff_id INT
AS
BEGIN
    SELECT
        l.id AS LessonID,
        l.booked_date AS LessonDate,
        l.start_time AS StartTime,
        l.end_time AS EndTime,
        l.price_group_name AS PriceGroupName,
        l.client_id AS ClientID,
        c.name AS ClientName,
        lp.used_car_number AS UsedCarNumber,
        lp.notes AS LessonNotes,
        lp.mileage AS LessonMileage,
        lp.mileage_fee AS MileageFee
    FROM
        Lesson l
        INNER JOIN
            Client c ON l.client_id = c.id
        LEFT JOIN
            LessonProgress lp ON l.id = lp.lesson_id
    WHERE
        l.instructor_staff_id = @Instructor_staff_id;
END;
GO;

EXEC GetInstructorLessons @Instructor_staff_id 3;

```

LessonID	LessonDate	StartTime	EndTime	PriceGroupName	ClientID	ClientName	UsedCarNumber	LessonNotes	LessonMileage	MileageFee
2	2012-12-21	11:02:51	12:02:56	Block	1002	Brad Pitt	CAR007	Good skills	24	37.00
4	2013-01-15	11:02:51	12:02:56	Block	1002	Brad Pitt	CAR007	Very well	24	37.00

2)



```

CREATE OR ALTER PROCEDURE GetInstructorLessonsByDateRange
    @Instructor_staff_id INT,
    @start_date DATE
AS
BEGIN
    SELECT
        l.id AS LessonID, l.booked_date AS LessonDate, l.start_time AS StartTime, l.end_time AS EndTime,
        l.price_group_name AS PriceGroupName,
        l.client_id AS ClientID,
        c.name AS ClientName,
        lp.used_car_number AS UsedCarNumber,
        lp.notes AS LessonNotes,
        lp.mileage AS LessonMileage,
        lp.mileage_fee AS MileageFee
    FROM
        Lesson l
        INNER JOIN
            Client c ON l.client_id = c.id
        LEFT JOIN
            LessonProgress lp ON l.id = lp.lesson_id
    WHERE
        l.instructor_staff_id = @Instructor_staff_id
        AND l.booked_date BETWEEN @start_date AND DATEADD(DAY, 7, @start_date);
END;
GO;

EXEC GetInstructorLessonsByDateRange @Instructor_staff_id 1, @start_date '2023-11-08'

```

LessonID	LessonDate	StartTime	EndTime	PriceGroupName	ClientID	ClientName	UsedCarNumber	LessonNotes	LessonMileage	MileageFee
1	2023-11-09	09:44:21	10:44:26	Individual	1	John Doe	CAR006	No experience	12	6.00
6	2023-11-14	13:34:15	14:34:17	Individual	2	Jane Smith	<null>	<null>	<null>	<null>

14

3)

The screenshot shows a SQL Server Enterprise Manager interface. The main window displays the execution of a stored procedure named `GetClientLessonsByDateRange`. The procedure is defined as follows:

```

1 use EasyDriveMotorSchool
2 GO;
3
4 CREATE OR ALTER PROCEDURE GetClientLessonsByDateRange
5     @client_id INT,
6     @start_date DATE
7 AS
8 BEGIN
9     SELECT
10         l.id AS LessonID,
11         l.booked_date AS LessonDate,
12         l.start_time AS StartTime,
13         l.end_time AS EndTime,
14         l.price_group_name AS PriceGroupName,
15         l.instructor_staff_id AS InstructorID,
16         s.name AS InstructorName,
17         lp.used_car_number AS UsedCarNumber,
18         lp.notes AS LessonNotes,
19         lp.mileage AS LessonMileage,
20         lp.mileage_fee AS MileageFee
21     FROM
22         Lesson l
23         INNER JOIN
24         Staff s ON l.instructor_staff_id = s.id
25         LEFT JOIN
26         LessonProgress lp ON l.id = lp.lesson_id
27     WHERE
28         l.client_id = @client_id
29         AND l.booked_date BETWEEN @start_date AND DATEADD(DAY, 7, @start_date);
30 END
31 GO;
32
33 EXEC GetClientLessonsByDateRange @client_id: 1, @start_date: '2023-11-08'
34 GO;

```

The output of the procedure is displayed in the 'Output' pane, showing a single row of data:

LessonID	LessonDate	StartTime	EndTime	Pri...	InstructorID	InstructorName	UsedCarNumber	LessonNotes	LessonMileage	MileageFee
1	2023-11-09	09:44:21	10:44:26	Individual	1	John Smith	CAR806	No experience	12	6.00

The right-hand pane shows the database structure, including tables, views, and routines.

4)

The screenshot shows a SQL Server Enterprise Manager interface. The main window displays the execution of a stored procedure named `GetInstructorAppointmentsNextWeek`. The procedure is defined as follows:

```

1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER PROCEDURE GetInstructorAppointmentsNextWeek
5     @instructor_id INT,
6     @date_requested DATE
7 AS
8 BEGIN
9     SELECT
10         CONCAT(l.booked_date, ' ', RIGHT(CONVERT(VARCHAR, l.booked_time, 100), 7)) as date_time,
11         c.id as client_id, c.name as client_name, 'Interview' as appointment_type
12     FROM EasyDriveMotorSchool.dbo.Interview i
13     INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = i.client_id
14     WHERE (i.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0) -- Start of next week
15           AND i.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND i.staff_id = @instructor_id -- Start of the week
16
17     UNION
18
19     SELECT
20         CONCAT(l.booked_date, ' ', RIGHT(CONVERT(VARCHAR, l.start_time, 100), 7)) as date_time,
21         c2.id as client_id, c2.name as client_name, 'Lesson' as appointment_type
22     FROM EasyDriveMotorSchool.dbo.Lesson l
23     INNER JOIN EasyDriveMotorSchool.dbo.Client c2 on c2.id = l.client_id
24     WHERE (l.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0) -- Start of next week
25           AND l.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND l.instructor_staff_id = @instructor_id -- Start of the week
26
27 HINTON
28
29 GetInstructorAppointmentsNextWeek()

```

The output of the procedure is displayed in the 'Output' pane, showing five rows of data:

date_time	client_id	client_name	appointment_type
2023-11-08 8:33AM	1	John Doe	Interview
2023-11-08 9:34AM	2	Jane Smith	Interview
2023-11-09 9:44AM	1	John Doe	Lesson
2023-11-11	1	John Doe	Driving Test
2023-11-12	1	John Doe	Driving Test

The right-hand pane shows the database structure, including tables, views, and routines.

15

VIEWS

Research how to make views in SQL server.

- 5) Create a View called Client_Lesson which does an inner join on the Client and Lesson tables. Run it to make sure it works properly!
- 6) Create a View called Lesson_Info which calls the View above Client_Lesson, and outputs all the information from Client_Lesson, along with who the staff person is for the lesson, i.e. the staff person's name and staffID.

One View can call another view, which makes things very flexible.

- 7) Create two more views that may be useful to you. Test them!

5)

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the 'EasyDriveMotorSchool' database with a table 'Client_Lesson' highlighted. The right pane shows the 'Database' tree with 'views' expanded, listing several views including 'FindInstructorWithoutCar', 'GetClientLessonDetails', 'GetClientLessonsByDateRange', 'GetInstructorAppointmentsNextWeek', 'GetInstructorLessons', 'GetInstructorLessonsByDateRange', 'GetTotalLessonsBeforeDate', and 'GetTotalLessonsForClient'.

The main pane shows the SQL script for creating the 'Client_Lesson' view:

```
1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER VIEW Client_Lesson AS
5 SELECT
6     l.id AS LessonID,
7     l.booked_date AS LessonDate,
8     l.start_time AS StartTime,
9     l.end_time AS EndTime,
10    l.price_group_name AS PriceGroupName,
11    l.client_id AS ClientID,
12    c.name AS ClientName,
13    c.gender AS ClientGender,
14    c.birth_date AS ClientBirthDate,
15    c.phone AS ClientPhone,
16    c.office_id AS ClientOfficeID,
17    lp.used_car_number AS UsedCarNumber
18 FROM
19     Lesson l
20     INNER JOIN
21     Client c ON l.client_id = c.id
22     LEFT JOIN LessonProgress LP
23     ON l.id = LP.Lesson_Id;
24 GO;
25
26 SELECT * FROM Client_Lesson;
```

The bottom pane shows the output of the 'Client_Lesson' view, displaying 6 rows of data:

LessonID	LessonDate	StartTime	EndTime	PriceGr...	ClientID	ClientName	ClientGender	ClientBirthDate	ClientPhone	ClientOfficeID	UsedCarNumber
1	2023-11-09	09:44:21	10:44:26	Individual	1	John Doe	M	1990-05-15	123-456-7890	1	CAR006
2	2012-12-21	11:02:51	12:02:56	Block	1002	Brad Pitt	M	1991-11-02	444-333-2222	2	CAR007
3	2012-12-21	11:02:51	12:02:56	Block	1003	Michael Jord...	M	1985-11-09	333-222-1111	1	CAR001
4	2013-01-15	11:02:51	12:02:56	Block	1002	Brad Pitt	M	1991-11-02	444-333-2222	2	CAR007
5	2013-01-15	11:02:51	12:02:56	Block	1003	Michael Jord...	M	1985-11-09	333-222-1111	1	CAR006
6	2023-11-14	13:34:15	14:34:17	Individual	2	Jane Smith	F	1985-08-22	987-654-3210	2	<null>

6)

```

5) client_lesson.sql 6) lesson_info.sql
1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER VIEW Lesson_Info AS
5 SELECT
6     cl.LessonID,
7     cl.LessonDate,
8     cl.StartTime,
9     cl.EndTime,
10    cl.PriceGroupName,
11    cl.ClientID,
12    cl.ClientName,
13    cl.ClientGender,
14    cl.ClientBirthDate,
15    cl.ClientPhone,
16    cl.ClientOfficeID,
17    cl.UsedCarNumber,
18    s.id AS InstructorID,
19    s.name AS InstructorName
20 FROM
21     Client_Lesson cl
22     INNER JOIN
23     Lesson l ON cl.LessonID = l.id
24     INNER JOIN
25     Staff s ON l.instructor_staff_id = s.id;
26 GO;
27
28 Lesson_Info

```

LessonID	LessonDate	StartTime	EndTime	PriceGroupName	ClientID	ClientName	ClientGender	ClientBirthDate	ClientPhone	ClientOfficeID
1	2023-11-09	09:44:21	10:44:26	Individual	1	John Doe	M	1990-05-15	123-456-7890	
2	2012-12-21	11:02:51	12:02:56	Block	1002	Brad Pitt	M	1991-11-02	444-333-2222	
3	2012-12-21	11:02:51	12:02:56	Block	1003	Michael Jordan	M	1985-11-09	333-222-1111	
4	2013-01-15	11:02:51	12:02:56	Block	1002	Brad Pitt	M	1991-11-02	444-333-2222	
5	2013-01-15	11:02:51	12:02:56	Block	1003	Michael Jordan	M	1985-11-09	333-222-1111	
6	2023-11-14	13:34:15	14:34:17	Individual	2	Jane Smith	F	1985-08-22	987-654-3210	

7.1)

```

7.1) total_price_for_each_lesson.sql
1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER VIEW TotalPriceForLesson
5 AS
6 SELECT l.*,
7     LP6.amount as sub_total_price,
8     LP.mileage_fee as mileage_fee,
9     CONCAT(CONVERT(NVARCHAR, (LP.mileage_fee+LP6.amount)), '$') as total_price
10 FROM Lesson l
11     INNER JOIN LessonProgress LP on l.id = LP.lesson_id
12     INNER JOIN LessonPriceGroup LP6 on LP6.group_name = l.price_group_name;
13 GO;
14
15 SELECT * FROM TotalPriceForLesson;

```

id	booked_date	start_time	end_time	price_group_name	instructor_staff_id	client_id	sub_total_price	mileage_fee	total_price
1	2023-11-09	09:44:21	10:44:26	Individual		1	120.00	6.00	126.00\$
2	2012-12-21	11:02:51	12:02:56	Block		1002	90.00	37.00	127.00\$
3	2012-12-21	11:02:51	12:02:56	Block		1003	90.00	37.00	127.00\$
4	2013-01-15	11:02:51	12:02:56	Block		1002	90.00	37.00	127.00\$
5	2013-01-15	11:02:51	12:02:56	Block		1003	90.00	37.00	127.00\$

17

7.2)

The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window titled '7.2) total_number_of_client_lessons.sql'. The query code is as follows:

```
1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER VIEW Client_Booked_Lessons
5 AS
6 SELECT
7     c.id, c.name, COUNT(l.id) as lessons_number
8 FROM Client c
9 LEFT JOIN Lesson L on c.id = L.client_id
10 GROUP BY c.id, c.name;
11 GO;
12
13 SELECT * FROM Client_Booked_Lessons ORDER BY lessons_number DESC;
14 GO;
```

The bottom pane shows the output of the query, displaying 13 rows of data in a table with columns 'id', 'name', and 'lessons_number'.

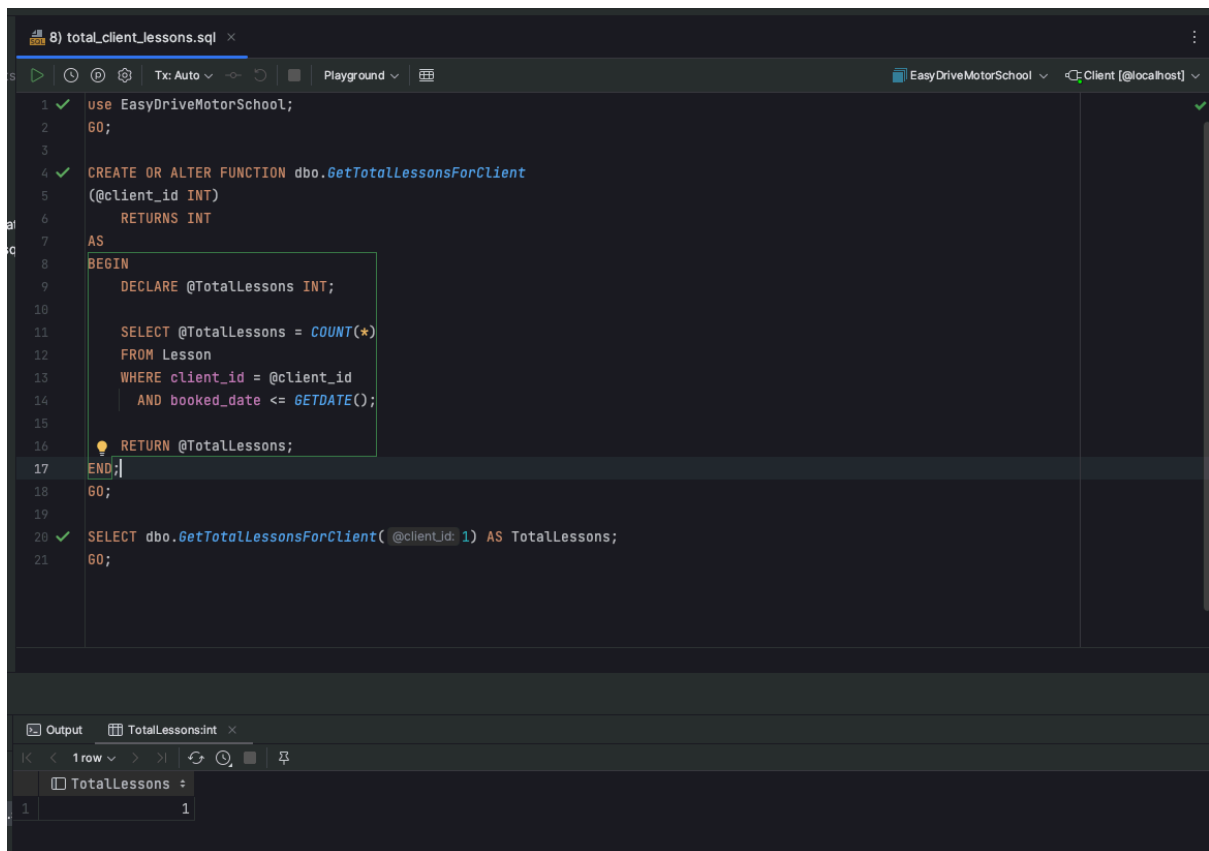
id	name	lessons_number
1002	Brad Pitt	2
1003	Michael Jordan	2
1	John Doe	1
2	Jane Smith	1
3	Alice Johnson	0
4	Bob Wilson	0
5	Eva Davis	0
6	Michael Brown	0
7	Sophia Taylor	0
8	David Lee	0
9	Olivia White	0

USER DEFINED FUNCTIONS

Research what user defined functions are and how to make them in SQL Server.

- 8) Create a user defined function that returns the total lessons that a client has taken up to today.
- 9) Create a user defined function that returns the total lessons that a client has taken before a date supplied by the user.
- 10) Create a user defined function that returns a table which does an inner join on the Client and Lesson tables, for a particular client which is supplied by the user. Run it to make sure it works properly!

8)



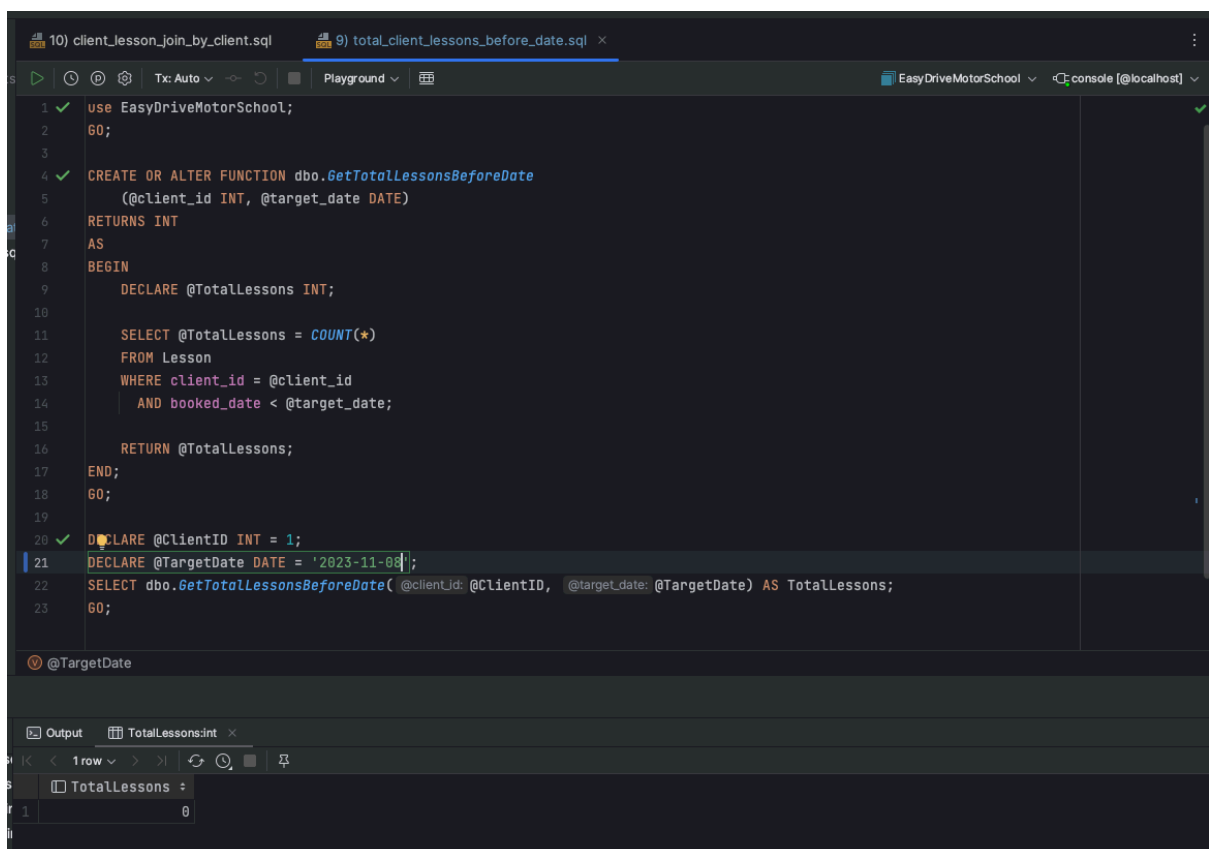
The screenshot shows a SQL Server Enterprise Manager window with a T-SQL script named '8) total_client_lessons.sql'. The script is executed against the 'EasyDriveMotorSchool' database. The script defines a function 'dbo.GetTotalLessonsForClient' that takes a client ID as input and returns the total number of lessons booked for that client up to the current date. The function is then used in a query to retrieve the total lessons for client ID 1.

```
1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER FUNCTION dbo.GetTotalLessonsForClient
5 (@client_id INT)
6 RETURNS INT
7 AS
8 BEGIN
9     DECLARE @TotalLessons INT;
10
11     SELECT @TotalLessons = COUNT(*)
12     FROM Lesson
13     WHERE client_id = @client_id
14     AND booked_date <= GETDATE();
15
16     RETURN @TotalLessons;
17 END;
18 GO;
19
20 SELECT dbo.GetTotalLessonsForClient( @clientId: 1) AS TotalLessons;
21 GO;
```

The output window shows the result of the query:

TotalLessons
1

9)



The screenshot shows a SQL Server Enterprise Manager window with a T-SQL script named '9) total_client_lessons_before_date.sql'. The script is executed against the 'EasyDriveMotorSchool' database. The script defines a function 'dbo.GetTotalLessonsBeforeDate' that takes a client ID and a target date as input and returns the total number of lessons booked for that client before the target date. The function is then used in a query to retrieve the total lessons for client ID 1 before the date '2023-11-08'.

```
1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER FUNCTION dbo.GetTotalLessonsBeforeDate
5 (@client_id INT, @target_date DATE)
6 RETURNS INT
7 AS
8 BEGIN
9     DECLARE @TotalLessons INT;
10
11     SELECT @TotalLessons = COUNT(*)
12     FROM Lesson
13     WHERE client_id = @client_id
14     AND booked_date < @target_date;
15
16     RETURN @TotalLessons;
17 END;
18 GO;
19
20 DECLARE @ClientID INT = 1;
21 DECLARE @TargetDate DATE = '2023-11-08';
22 SELECT dbo.GetTotalLessonsBeforeDate( @clientId: @ClientID, @target_date: @TargetDate) AS TotalLessons;
23 GO;
```

The output window shows the result of the query:

TotalLessons
0

10)

The screenshot shows a SQL IDE with a query editor on the left, a database explorer on the right, and an output window at the bottom. The query editor contains the following SQL code:

```

1 use EasyDriveMotorSchool;
2 GO;
3 CREATE OR ALTER FUNCTION dbo.GetClientLessonDetails
4
5 (@client_id INT)
6 RETURNS TABLE
7 AS
8 RETURN
9 (
10 SELECT
11     l.id,
12     booked_date,
13     start_time,
14     end_time,
15     price_group_name,
16     instructor_staff_id,
17     client_id,
18     name,
19     gender,
20     birth_date,
21     phone,
22     office_id
23 FROM Lesson l
24 JOIN Client c ON l.client_id = c.id
25 WHERE c.id = @client_id
26 )
27
28 GO;
29
30 GetClientLessonDetails()

```

The database explorer on the right shows the 'EasyDriveMotorSchool' database with various tables and views. The output window at the bottom displays the result set of the function call, showing a single row of data for client ID 1.

id	booked_date	start_time	end_time	price_group_name	instructor_staff_id	client_id	name	gender	birth_date	phone	office_id
1	2023-11-09	09:44:21	10:44:26	Individual	1	1	John Doe	M	1990-05-15	123-456-7890	

TRIGGERS

Research what triggers are and how to make them in SQL Server.

- 11) In the Staff table, add an attribute to keep track of the total number of clients that an instructor has. Whenever a new client is added to the Client table, we add one to the above new attribute, to the staff person who is working with this new client. A similar thing is done if a client is removed from our Client table.

11)

The screenshot shows a SQL IDE with a query editor containing the following SQL code:

```

1 use EasyDriveMotorSchool;
2 GO;
3
4 CREATE OR ALTER TRIGGER trg_ClientAdded
5 ON Interview
6 AFTER INSERT
7 AS
8 BEGIN
9     UPDATE Staff
10     SET total_clients_served = total_clients_served + 1
11     FROM Staff s
12     INNER JOIN inserted i ON s.id = i.staff_id;
13 END;
14 GO;
15
16 CREATE OR ALTER TRIGGER trg_ClientRemoved
17 ON Interview
18 AFTER DELETE
19 AS
20 BEGIN
21     UPDATE Staff
22     SET total_clients_served = IIF(total_clients_served > 0, total_clients_served - 1, 0)
23     FROM Staff s
24     INNER JOIN deleted d ON s.id = d.staff_id;
25 END;
26 GO;

```

The query editor shows the creation of two triggers: 'trg_ClientAdded' and 'trg_ClientRemoved'. The 'trg_ClientAdded' trigger updates the 'total_clients_served' attribute in the 'Staff' table when a new client is added to the 'Client' table. The 'trg_ClientRemoved' trigger updates the 'total_clients_served' attribute in the 'Staff' table when a client is removed from the 'Client' table.

20

CURSORS

Research what a cursor is and how to make them in SQL Server.

- 12) Use a cursor to read the rows of the Lesson table.
- If the mileage for the lesson was over 20 miles, increase the fee by \$5.
 - If the mileage for the lesson was over 25 miles, increase the fee by \$8.
 - If the mileage for the lesson was over 30 miles, increase the fee by \$10.

You can use an If ... ELSE ... statement.

- 13) Do the same thing as question 12, but now use a Case statement.

12)

The screenshot displays the SQL Server Enterprise Manager interface. The main window shows a T-SQL script titled '12) mileage_cursor_if.sql'. The script uses a cursor to iterate through the 'Lesson' table, updating the 'mileage_fee' based on the 'mileage' value. The 'Database' pane on the right shows the 'EasyDriveMotorSchool' database with various tables and routines. The 'Output' pane at the bottom shows the results of the script, displaying the updated 'LessonProgress' table.

```
1 use EasyDriveMotorSchool;
2 GO;
3
4
5 DECLARE @LessonID INT;
6 DECLARE @Mileage INT;
7 DECLARE @Fee DECIMAL(10, 2);
8
9
10 DECLARE LessonCursor CURSOR FOR
11     SELECT id, mileage, mileage_fee
12     FROM Lesson
13     INNER JOIN LessonProgress LP on Lesson.id = LP.lesson_id
14
15 OPEN LessonCursor;
16
17
18 FETCH NEXT FROM LessonCursor INTO @LessonID, @Mileage, @Fee;
19
20
21 WHILE @@FETCH_STATUS = 0
22 BEGIN
23     IF @Mileage > 30
24         SET @Fee = @Fee + 10;
25     ELSE IF @Mileage > 25
26         SET @Fee = @Fee + 8;
27     ELSE IF @Mileage > 20
28         SET @Fee = @Fee + 5;
29
30     UPDATE LessonProgress
```

lesson_id	notes	mileage	mileage_fee	used_car_number
1	No experience	12	6.00	CAR006
2	Good skills	24	42.00	CAR007
3	Nice skills	24	42.00	CAR001
4	Very well	24	42.00	CAR007
5	Good	24	42.00	CAR006

13)

The screenshot displays the SQL Server Enterprise Manager interface. The main window shows a T-SQL script titled '13) mileage_cursor_case.sql'. The script is as follows:

```
1 use EasyDriveMotorSchool;
2 GO;
3
4 DECLARE @LessonID INT;
5 DECLARE @Mileage INT;
6 DECLARE @Fee DECIMAL(10, 2);
7
8
9 DECLARE LessonCursor CURSOR FOR
10     SELECT id, mileage, mileage_fee
11     FROM Lesson
12     INNER JOIN LessonProgress LP on Lesson.id = LP.lesson_id
13
14 OPEN LessonCursor;
15
16
17 FETCH NEXT FROM LessonCursor INTO @LessonID, @Mileage, @Fee;
18
19
20 WHILE @@FETCH_STATUS = 0
21 BEGIN
22     -- Use CASE statement to update fee based on mileage conditions
23     SET @Fee =
24         CASE
25             WHEN @Mileage > 30 THEN @Fee + 10
26             WHEN @Mileage > 25 THEN @Fee + 8
27             WHEN @Mileage > 20 THEN @Fee + 5
28             ELSE @Fee
29         END;
30
```

The right-hand pane shows the 'Database' explorer with the 'EasyDriveMotorSchool' database selected. The 'dbo' schema is expanded, showing various tables and views. The 'LessonProgress' table is highlighted.

The bottom pane shows the 'Output' window with the query results for the 'LessonProgress' table. The results are as follows:

lesson_id	notes	mileage	mileage_fee	used_car_number
1	No experience	12	6.00	CAR006
2	Good skills	24	52.00	CAR007
3	Nice skills	24	52.00	CAR001
4	Very well	24	52.00	CAR007
5	Good	24	52.00	CAR006