Wednesday, 22 November 2023

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 indi- vidual 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.

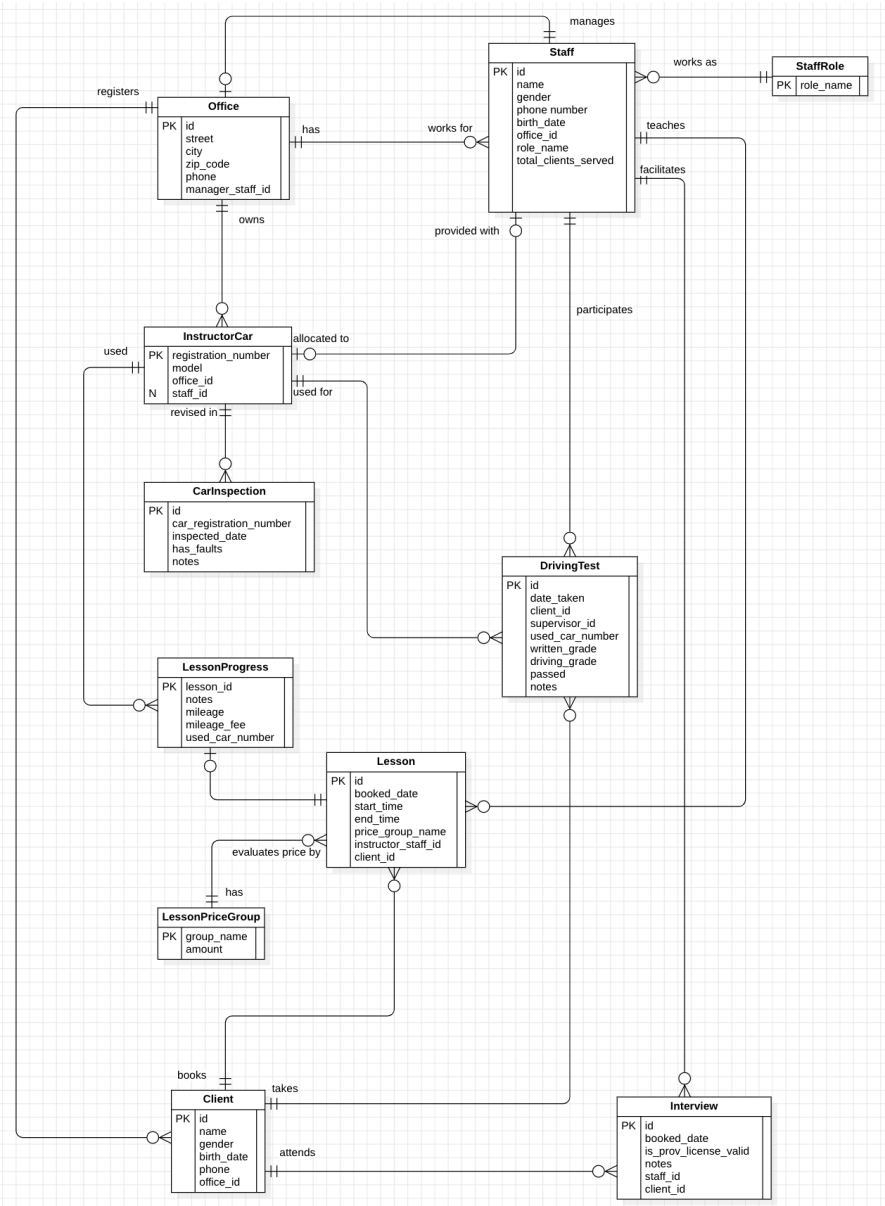
(l) The names of clients who passed the driving test in January 2013.

(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

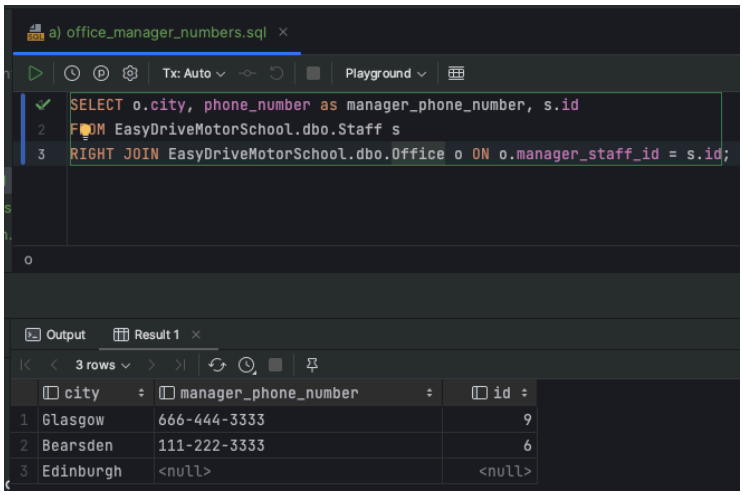
(used_car_number) -> supervisor_id

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:

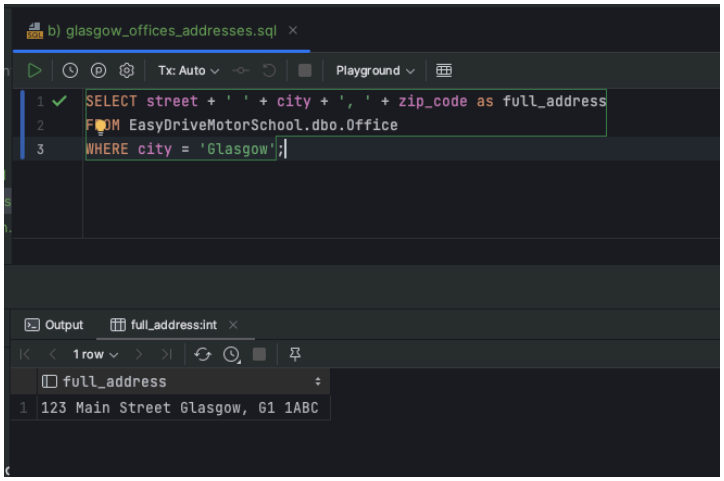
1. During either a lesson or driving test, instructor of “supervisor_id” may not provide own assigned car because, as an example, an office may have number of instructors greater than number of cars so, for a purpose of a lesson or a driving test, any other available car will be used.
2. A car of an 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:

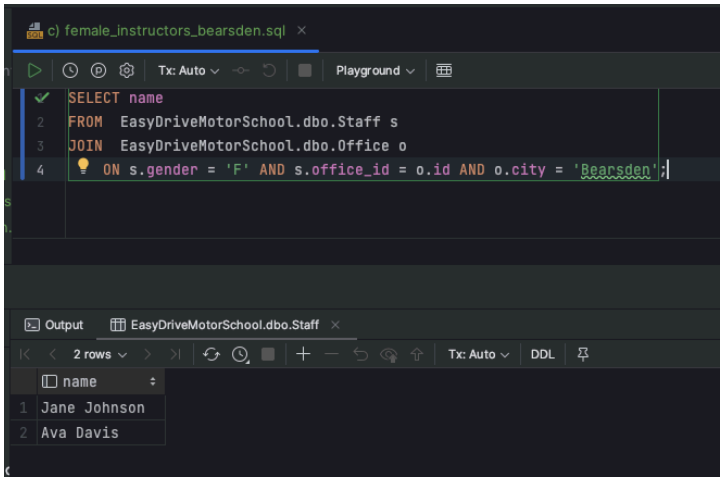
(a) The names and the telephone numbers of the Managers of each office.



(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.

d) office_staff_number.sql

```sql
SELECT
    o.street + ' ' + o.city as office,
    (
        SELECT COUNT(*)
        FROM EasyDriveMotorSchool.dbo.Staff s
        WHERE s.office_id = o.id
    ) as staff_number
FROM EasyDriveMotorSchool.dbo.Office o;
```

staff_number

Output | Result 1

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.

e) clients_per_city.sql

```sql
SELECT
    city,
    COUNT(c.office_id) as clients_number
FROM EasyDriveMotorSchool.dbo.Office o
LEFT JOIN EasyDriveMotorSchool.dbo.Client c ON c.office_id = o.id
GROUP BY city;
```

Output | Result 1

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.

```sql
use EasyDriveMotorSchool;
GO;

CREATE OR ALTER PROCEDURE InstructorAppointmentsNextWeek
    @instructor_id INT,
    @date_requested DATE
as
BEGIN
    SELECT
    CONCAT(i.booked_date, ' ', RIGHT(CONVERT(VARCHAR, i.booked_time, 100), 7)) as date_time,
    c.id as client_id, c.name as client_name, 'Interview' as appointment_type
    FROM EasyDriveMotorSchool.dbo.Interview i
    INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = i.client_id
    WHERE (i.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0)  -- Start of next week
      AND i.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND i.staff_id = @instructor_id -- Start of the week after next

    UNION

    SELECT
        CONCAT(l.booked_date, ' ', RIGHT(CONVERT(VARCHAR, l.start_time, 100), 7)) as date_time,
        c2.id as client_id, c2.name as client_name, 'Lesson' as appointment_type
    FROM EasyDriveMotorSchool.dbo.Lesson l
    INNER JOIN EasyDriveMotorSchool.dbo.Client c2 on C2.id = l.client_id
    WHERE (l.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0)  -- Start of next week
        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

    UNION

    SELECT
        t.date_taken as date_time,
        c2.id as client_id, c2.name as client_name, 'Driving Test' as appointment_type
    FROM EasyDriveMotorSchool.dbo.DrivingTest t
            INNER JOIN EasyDriveMotorSchool.dbo.Client c2 on C2.id = t.client_id
    WHERE (t.date_taken >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0)  -- Start of next week
        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
END
GO;

EXEC InstructorAppointmentsNextWeek @instructor_id = 1, @date_requested = '2023-11-1';
```

Services - easy_drive_motor_school

Services

5 rows

	date_time	client_id	client_name	appointment_type
1	2023-11-08	1	John Doe	Interview
2	2023-11-08	2	Jane Smith	Interview
3	2023-11-09	1	John Doe	Lesson
4	2023-11-11	1	John Doe	Driving Test
5	2023-11-12	1	John Doe	Driving Test

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

```sql
use EasyDriveMotorSchool;
GO;

CREATE OR ALTER PROCEDURE InstructorInterviewDetails
    @instructor_id INT
as
BEGIN
    SELECT
        CONCAT(i.booked_date, ' ', RIGHT(CONVERT(VARCHAR, i.booked_time, 100), 7)) as date_time,
        c.id as client_id, c.name as client_name, i.notes, i.is_prov_license_valid
    FROM EasyDriveMotorSchool.dbo.Interview i
            INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = i.client_id
    WHERE i.staff_id = @instructor_id
END
GO;

EXEC InstructorInterviewDetails @instructor_id = 1;
```

2 rows

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

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

Output | Result 1-2 | Result 1-3

2 rows

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

```sql
SELECT s.name, s.phone_number, DATEDIFF(YEAR, s.birth_date, GETDATE()) as age
FROM EasyDriveMotorSchool.dbo.Staff s
WHERE (role_name = 'Instructor' OR role_name = 'Senior Instructor') AND DATEDIFF(YEAR, s.birth_date, GETDATE()) >= 55;
```

Output | Result 1

5 rows

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

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

j) reg_number_cars_with_no_faults.sql

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

Output | EasyDriveMotorSchool.dbo.InstructorCar

3 rows

registration_number
CAR002
CAR003
CAR004

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

k) cars_used_by_instructors_bearsden.sql

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

Output | EasyDriveMotorSchool.dbo.InstructorCar

5 rows

registration_number
CAR001
CAR002
CAR003
CAR004
CAR005

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

l) name_clients_passed_test_jan2013.sql

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

Output | EasyDriveMotorSchool.dbo.Client

2 rows

name
Brad Pitt
Michael Jordan

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

m) failed_driving_test_more_than_3_times_havent_passed.sql

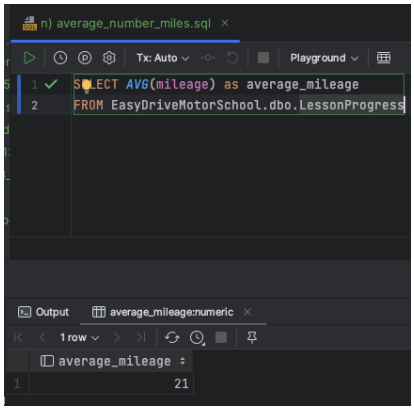
```sql
SELECT  c.name
FROM EasyDriveMotorSchool.dbo.DrivingTest t
INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = t.client_id
GROUP BY c.name, c.id, t.passed
HAVING COUNT(CASE WHEN t.passed = 0 THEN 1 END) > 3 AND COUNT(CASE WHEN t.passed = 1 THEN 1 END) = 0
```

Output | EasyDriveMotorSchool.dbo.Client

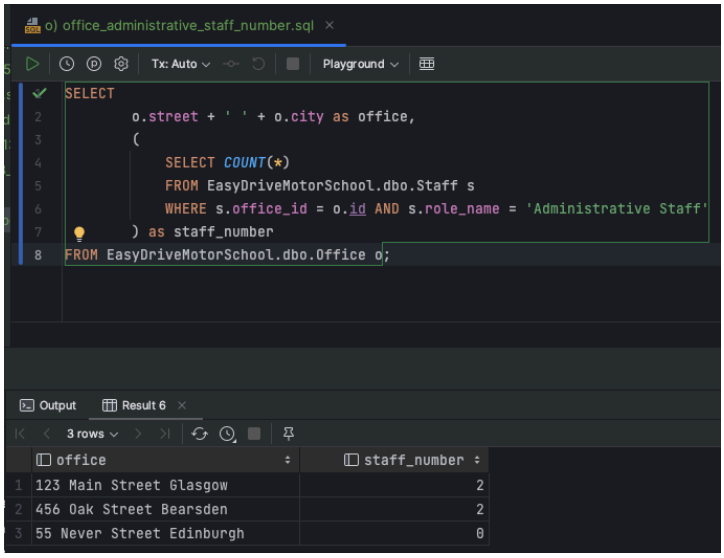
1 row

name
John Doe

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



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



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)

```sql
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)

```sql
use EasyDriveMotorSchool;
GO;

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>

3)

3) display_lessons_of_client_in_7_days_range.sql

```sql
use EasyDriveMotorSchool
GO;

CREATE OR ALTER PROCEDURE GetClientLessonsByDateRange
    @client_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.instructor_staff_id AS InstructorID,
        s.name AS InstructorName,
        lp.used_car_number AS UsedCarNumber,
        lp.notes AS LessonNotes,
        lp.mileage AS LessonMileage,
        lp.mileage_fee AS MileageFee
    FROM
        Lesson l
            INNER JOIN
        Staff s ON l.instructor_staff_id = s.id
            LEFT JOIN
        LessonProgress lp ON l.id = lp.lesson_id
    WHERE
            l.client_id = @client_id
      AND l.booked_date BETWEEN @start_date AND DATEADD(DAY, 7, @start_date);
END
GO;

EXEC GetClientLessonsByDateRange @client_id: 1, @start_date: '2023-11-08'
GO;
```

GetClientLessonsByDateRange()

Output | Result 1-3

1 row

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	CAR006	No experience	12	6.00

4)

4) display_instructor_appointments_next_week.sql

```sql
use EasyDriveMotorSchool;
GO;

CREATE OR ALTER PROCEDURE GetInstructorAppointmentsNextWeek
    @instructor_id INT,
    @date_requested DATE
as
BEGIN
    SELECT
    CONCAT(i.booked_date, ' ', RIGHT(CONVERT(VARCHAR, i.booked_time, 100), 7)) as date_time,
    c.id as client_id, c.name as client_name, 'Interview' as appointment_type
    FROM EasyDriveMotorSchool.dbo.Interview i
    INNER JOIN EasyDriveMotorSchool.dbo.Client c on c.id = i.client_id
    WHERE (i.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0)  -- Start of next week
      AND i.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND i.staff_id = @instructor_id -- Start of the week

    UNION

    SELECT
        CONCAT(l.booked_date, ' ', RIGHT(CONVERT(VARCHAR, l.start_time, 100), 7)) as date_time,
        c2.id as client_id, c2.name as client_name, 'Lesson' as appointment_type
    FROM EasyDriveMotorSchool.dbo.Lesson l
    INNER JOIN EasyDriveMotorSchool.dbo.Client c2 on C2.id = l.client_id
    WHERE (l.booked_date >= DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 1, 0)  -- Start of next week
        AND l.booked_date < DATEADD(WEEK, DATEDIFF(WEEK, 0, @date_requested) + 2, 0)) AND l.instructor_staff_id = @instructor_id -- Star

    UNION
```

GetInstructorAppointmentsNextWeek()

Output | Result 1-3

5 rows

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

VIEWS

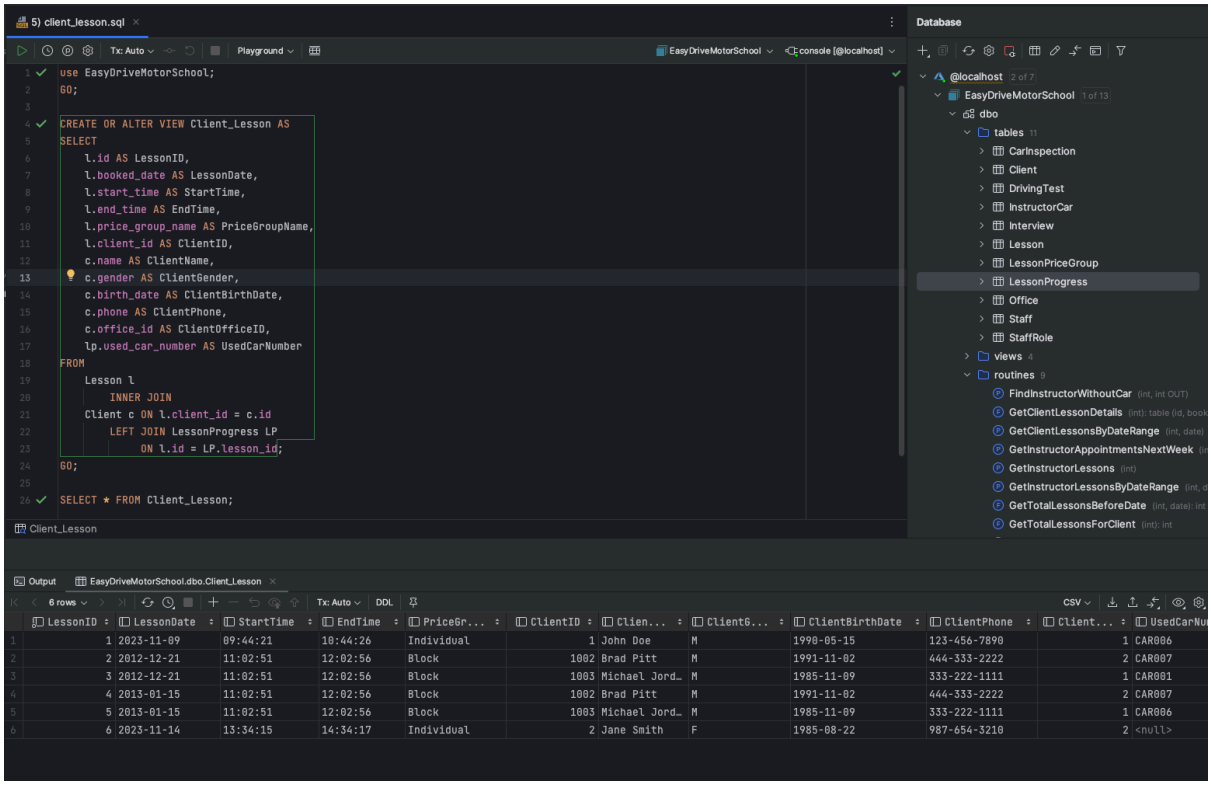
Research how to make views in SQL server.

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)



6)

```sql
use EasyDriveMotorSchool;
GO;

CREATE or ALTER VIEW Lesson_Info AS
SELECT
    cl.LessonID,
    cl.LessonDate,
    cl.StartTime,
    cl.EndTime,
    cl.PriceGroupName,
    cl.ClientID,
    cl.ClientName,
    cl.ClientGender,
    cl.ClientBirthDate,
    cl.ClientPhone,
    cl.ClientOfficeID,
    cl.UsedCarNumber,
    s.id AS InstructorID,
    s.name AS InstructorName
FROM
    Client_Lesson cl
        INNER JOIN
    Lesson l ON cl.LessonID = l.id
        INNER JOIN
    Staff s ON l.instructor_staff_id = s.id;
GO;
```

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)

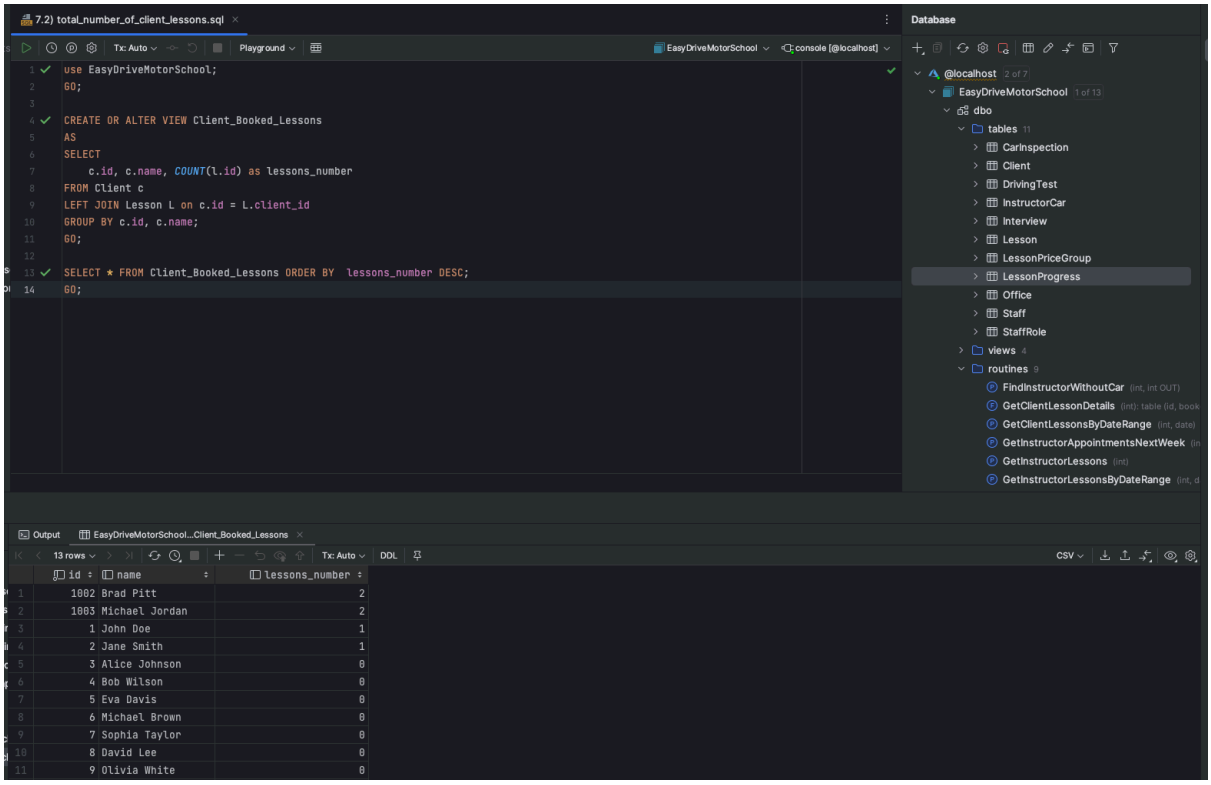
```sql
use EasyDriveMotorSchool;
GO;

CREATE OR ALTER VIEW TotalPriceForLesson
AS
SELECT  l.*,
        LPG.amount as sub_total_price,
        LP.mileage_fee as mileage_fee,
        CONCAT(CONVERT(NVARCHAR, (LP.mileage_fee+LPG.amount)), '$') as total_price
FROM Lesson l
INNER JOIN LessonProgress LP on l.id = LP.lesson_id
INNER JOIN LessonPriceGroup LPG on LPG.group_name = l.price_group_name;
GO;

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	1	120.00	6.00	126.00$
2	2012-12-21	11:02:51	12:02:56	Block	3	1002	90.00	37.00	127.00$
3	2012-12-21	11:02:51	12:02:56	Block	4	1003	90.00	37.00	127.00$
4	2013-01-15	11:02:51	12:02:56	Block	3	1002	90.00	37.00	127.00$
5	2013-01-15	11:02:51	12:02:56	Block	5	1003	90.00	37.00	127.00$

7.2)



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)

8) total_client_lessons.sql

```sql
use EasyDriveMotorSchool;
GO;

CREATE OR ALTER FUNCTION dbo.GetTotalLessonsForClient
(@client_id INT)
    RETURNS INT
AS
BEGIN
    DECLARE @TotalLessons INT;

    SELECT @TotalLessons = COUNT(*)
    FROM Lesson
    WHERE client_id = @client_id
      AND booked_date <= GETDATE();

    RETURN @TotalLessons;
END;
GO;

SELECT dbo.GetTotalLessonsForClient( @client_id: 1) AS TotalLessons;
GO;
```

Output | TotalLessons:int

1 row

	TotalLessons
1	1

9)

10) client_lesson_join_by_client.sql | 9) total_client_lessons_before_date.sql

```sql
use EasyDriveMotorSchool;
GO;

CREATE OR ALTER FUNCTION dbo.GetTotalLessonsBeforeDate
    (@client_id INT, @target_date DATE)
RETURNS INT
AS
BEGIN
    DECLARE @TotalLessons INT;

    SELECT @TotalLessons = COUNT(*)
    FROM Lesson
    WHERE client_id = @client_id
      AND booked_date < @target_date;

    RETURN @TotalLessons;
END;
GO;

DECLARE @ClientID INT = 1;
DECLARE @TargetDate DATE = '2023-11-08';
SELECT dbo.GetTotalLessonsBeforeDate( @client_id: @ClientID, @target_date: @TargetDate) AS TotalLessons;
GO;
```

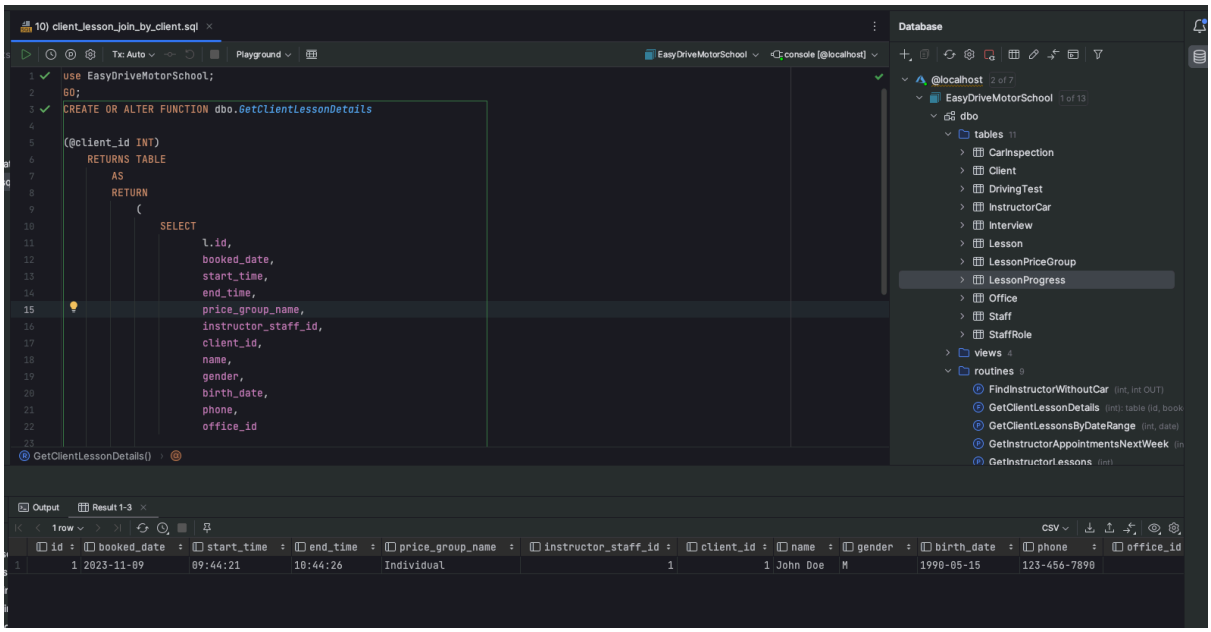
@TargetDate

Output | TotalLessons:int

1 row

	TotalLessons
1	0

10)

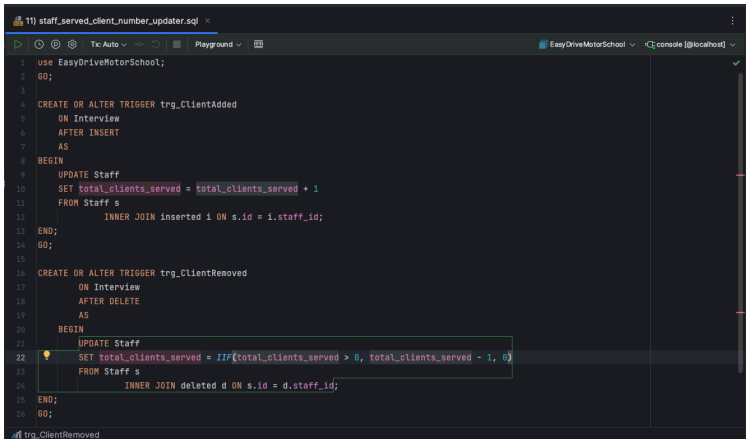


TRIGGERS

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)



CURSOR

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

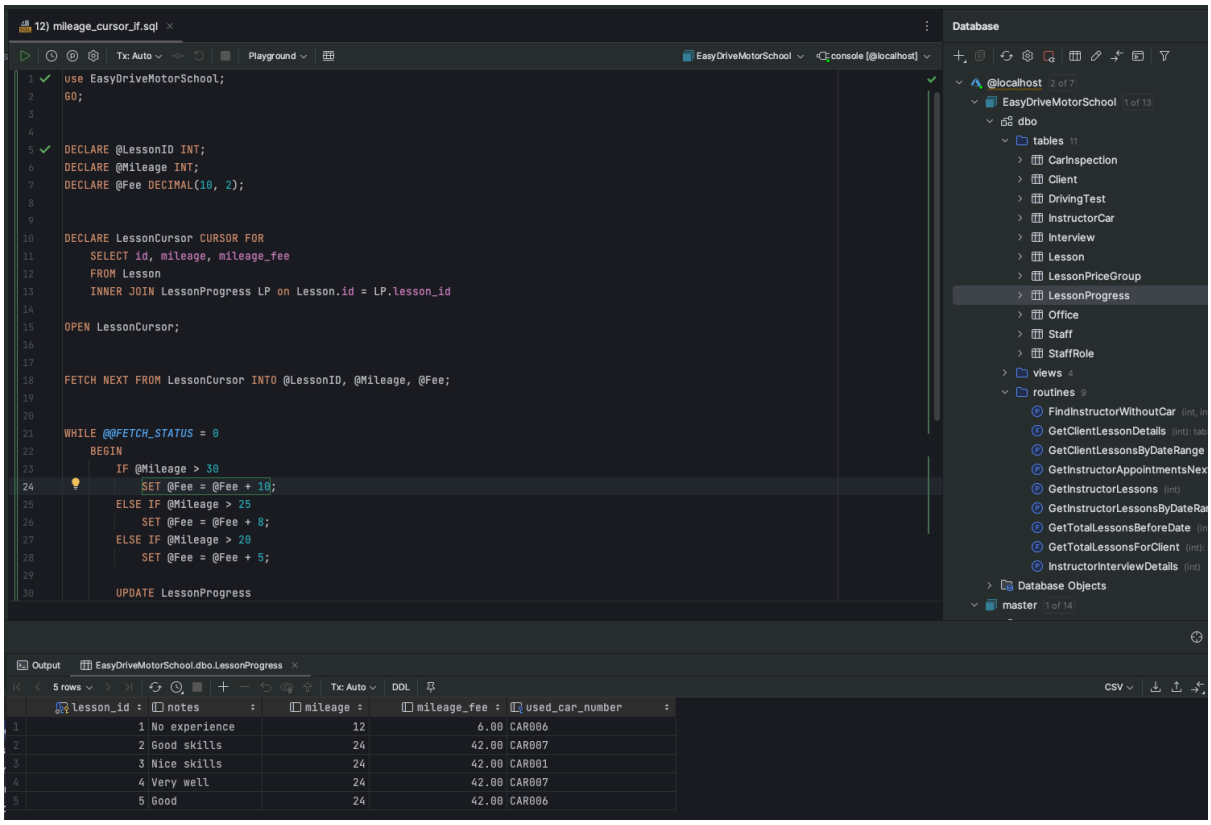
12) Use a cursor to read the rows of the Lesson table.
    - If the mileage for the lesson was over 20 miles, increase the fee by $5.
    - If the mileage for the lesson was over 25 miles, increase the fee by $8.
    - If the mileage for the lesson was over 30 miles, increase the fee by $10.

    You can use an If … ELSE … statement.

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

---

12)



13)

13) mileage_cursor_case.sql

```sql
use EasyDriveMotorSchool;
GO;

DECLARE @LessonID INT;
DECLARE @Mileage INT;
DECLARE @Fee DECIMAL(10, 2);


DECLARE LessonCursor CURSOR FOR
    SELECT id, mileage, mileage_fee
    FROM Lesson
            INNER JOIN LessonProgress LP on Lesson.id = LP.lesson_id

OPEN LessonCursor;


FETCH NEXT FROM LessonCursor INTO @LessonID, @Mileage, @Fee;


WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Use CASE statement to update fee based on mileage conditions
        SET @Fee =
                CASE
                    WHEN @Mileage > 30 THEN @Fee + 10
                    WHEN @Mileage > 25 THEN @Fee + 8
                    WHEN @Mileage > 20 THEN @Fee + 5
                    ELSE @Fee
                    END;
```

Output | EasyDriveMotorSchool.dbo.LessonProgress

5 rows

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