**EZEKIEL ISAAC NARAYANASAMI**

**ST10359128**

**INSY7213**

**PRACTICAL ASSIGNMENT 1**

**QUESTION 1**

**1 .. M**

**1 .. M**

|  |
| --- |
| CUSTOMER |
| PK CustomerID  FirstName  Surname  Address  PhoneNumber  Email |

|  |
| --- |
| BILLING |
| PK BillID  FK CustomerID  FK StaffID  BillDate |

|  |
| --- |
| STAFF |
| PK StaffID  FirstName  Surname  Position  PhoneNumber  Address  Email |

|  |
| --- |
| DELIVERY ITEMS |
| PK DeliveryItem  Description  FK StaffID |

**1 .. M**

|  |
| --- |
| DRIVER DELIVERIES |
| PK DriverDelivID  FK DeliveryItem  FK DriverID  FK VINNumber |

**1 .. M**

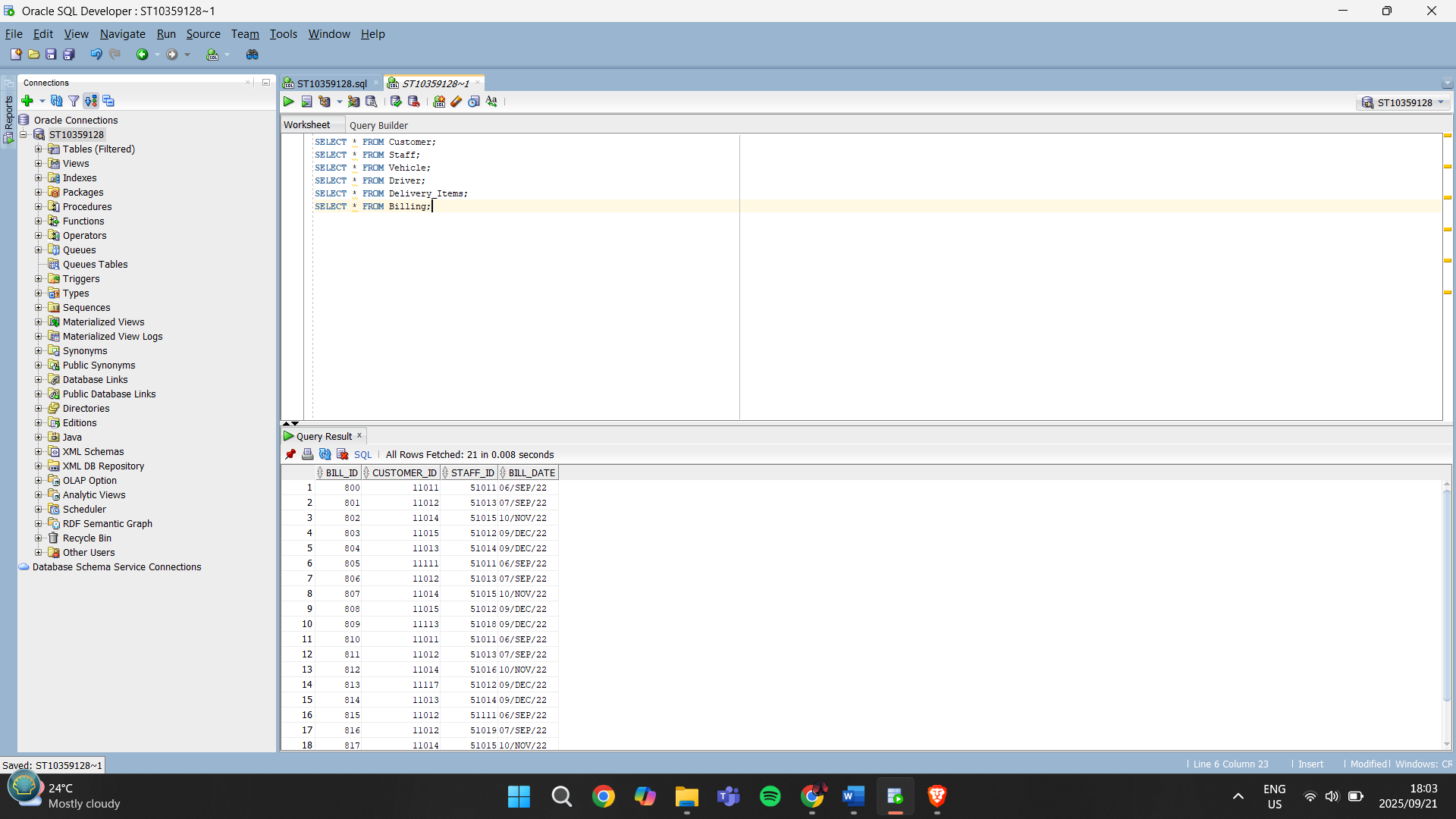
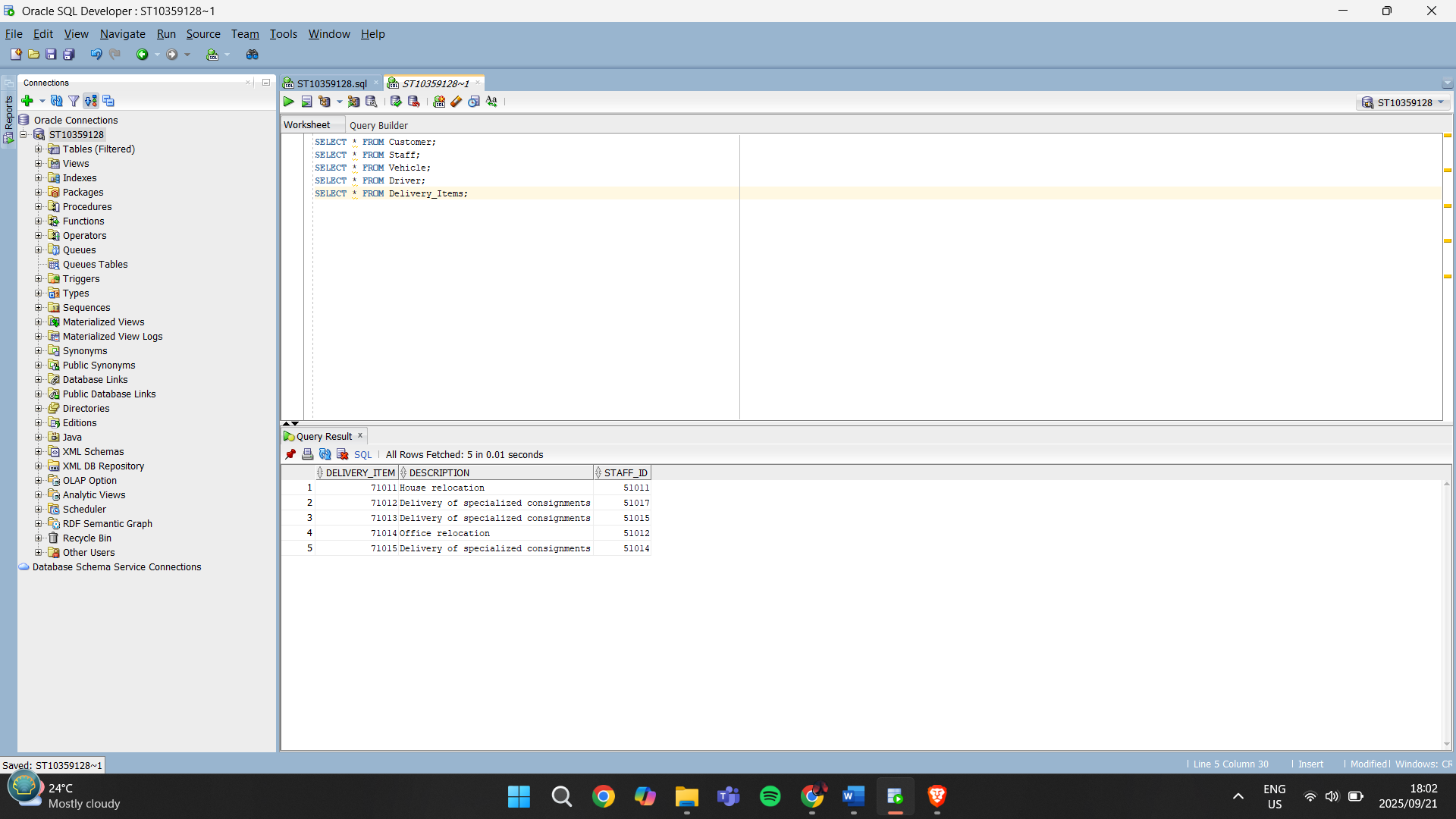
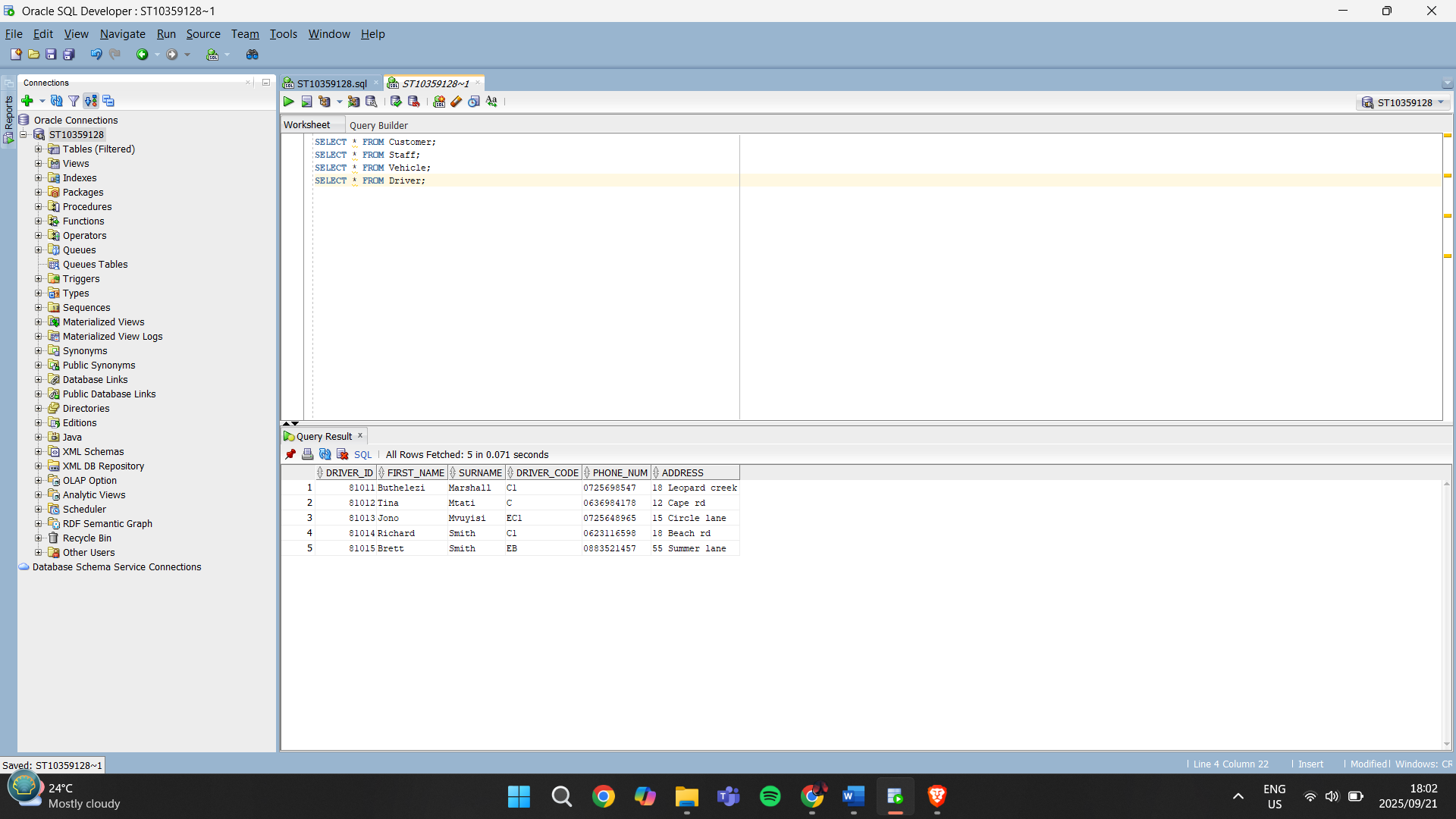
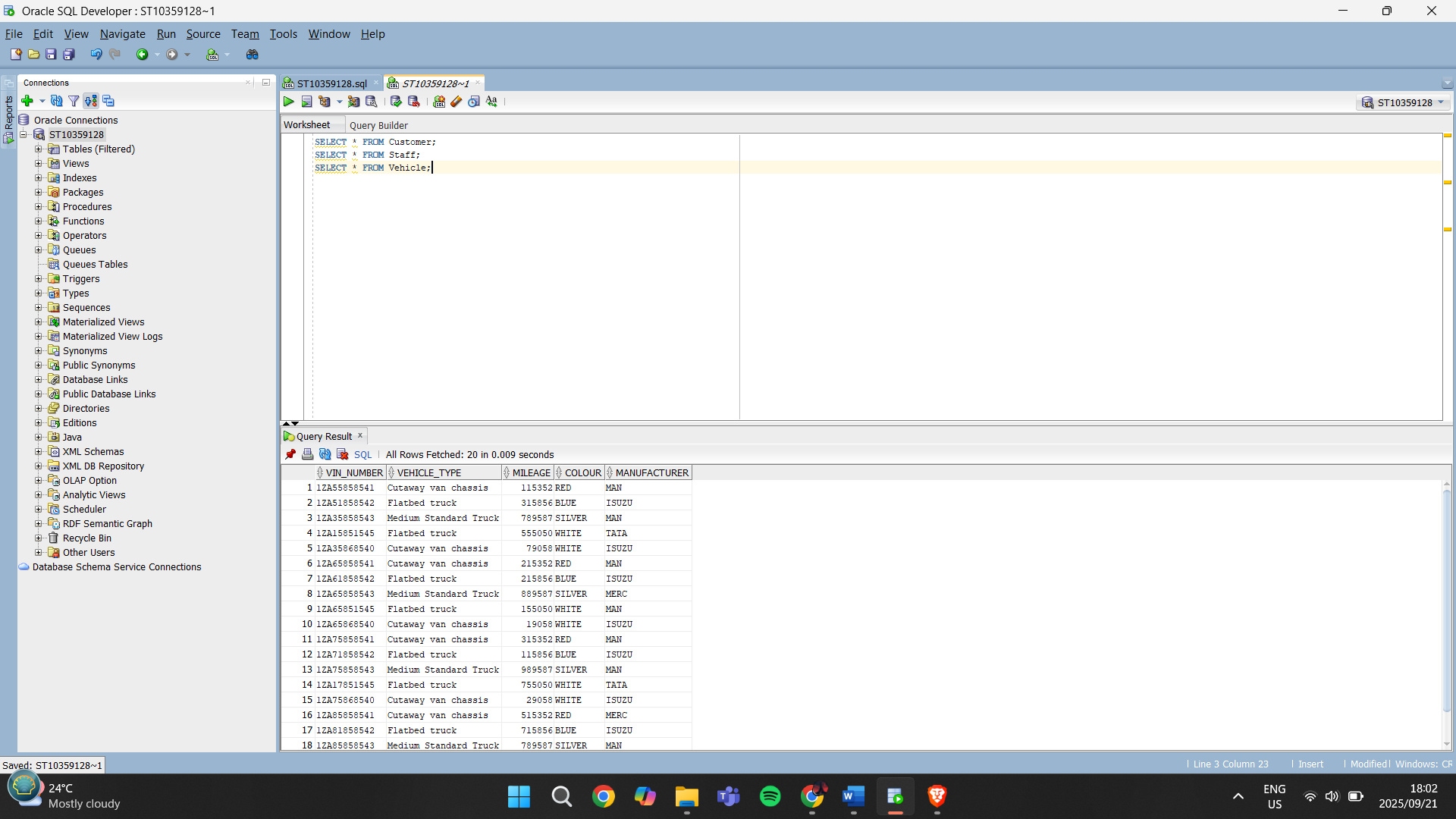
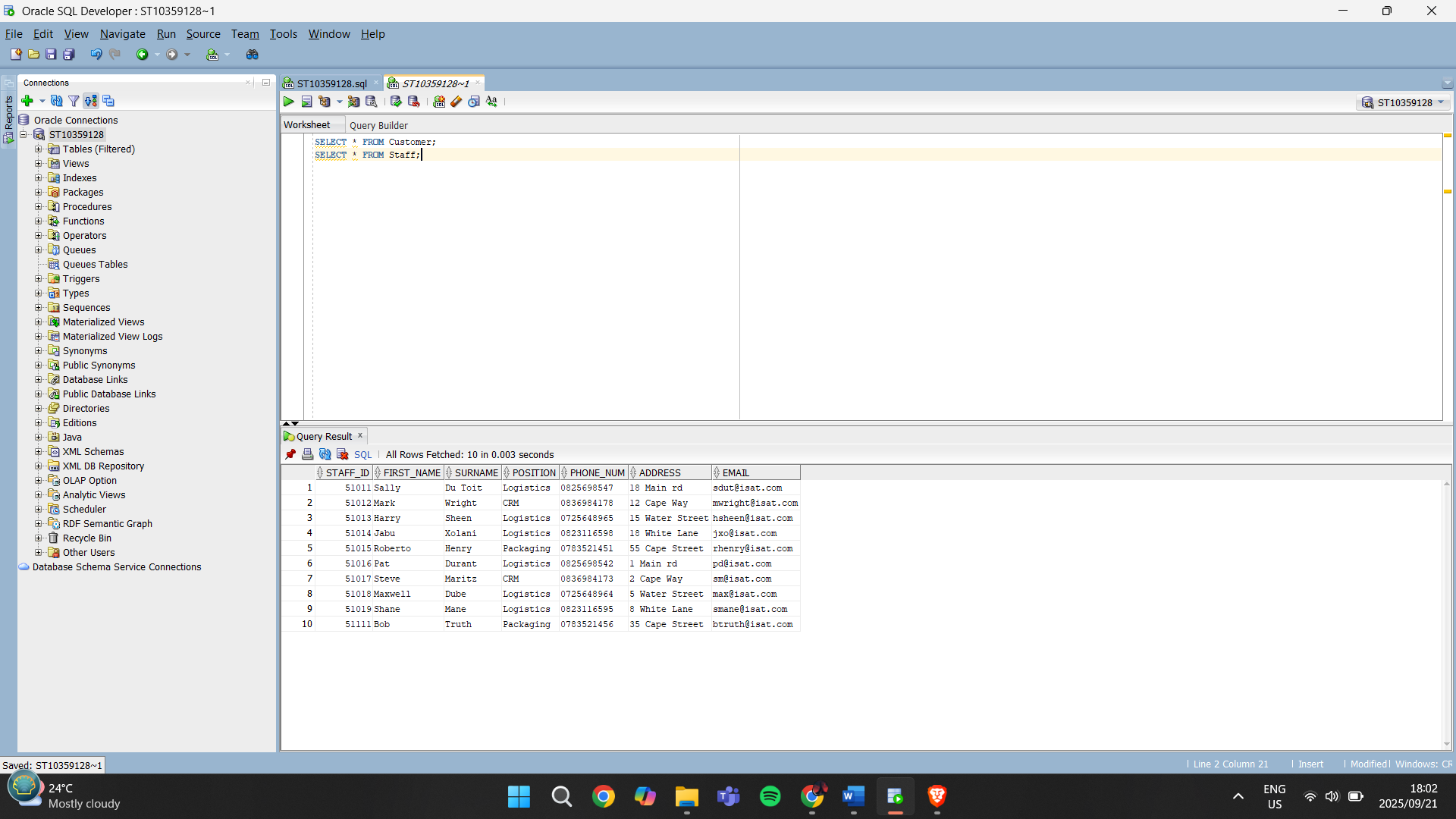
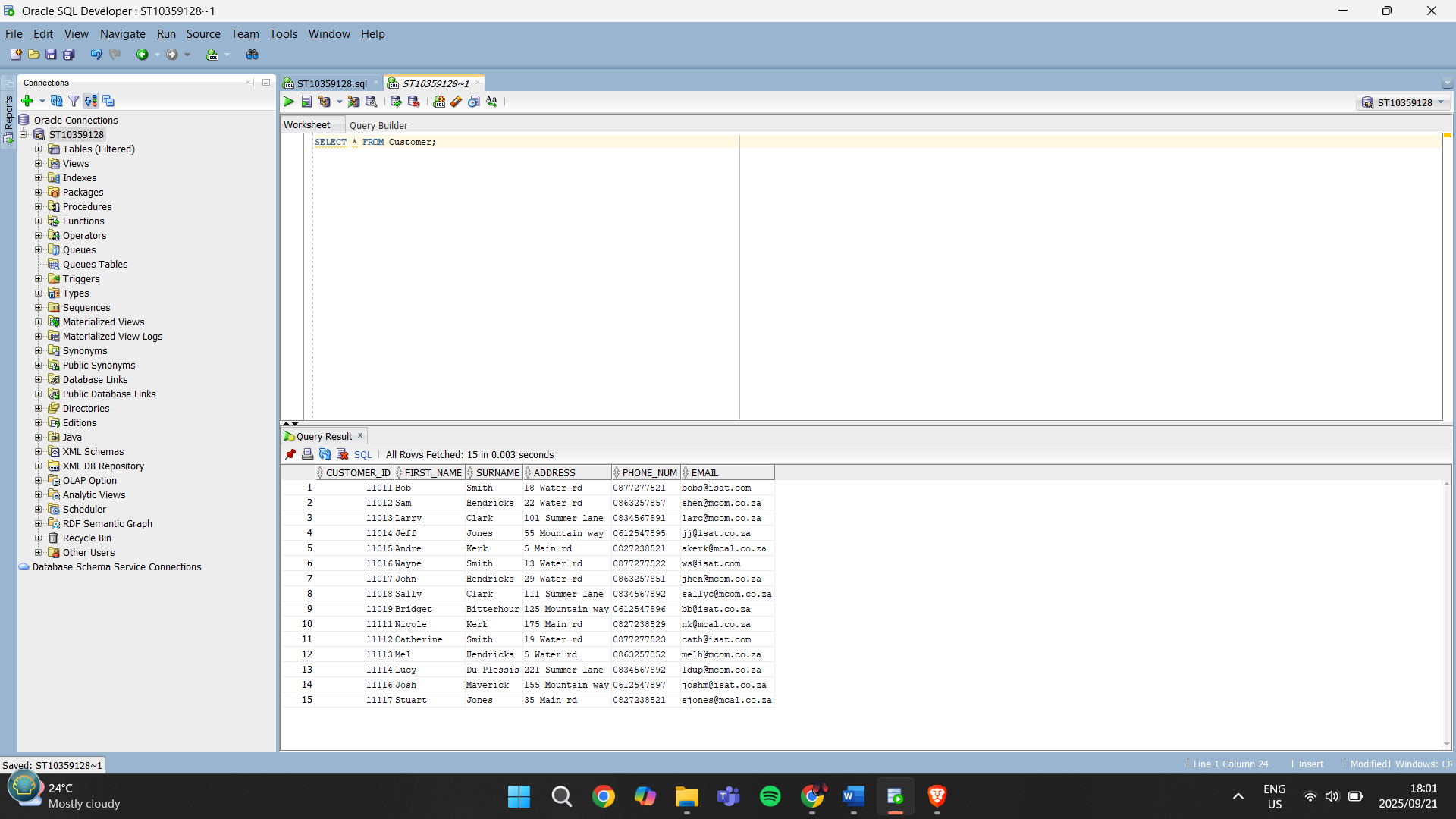
|  |
| --- |
| VEHICLE |
| PK VINNumber  VehicleType  Mileage  Colour  Manufacturer |

**M .. 1**

**M .. 1**

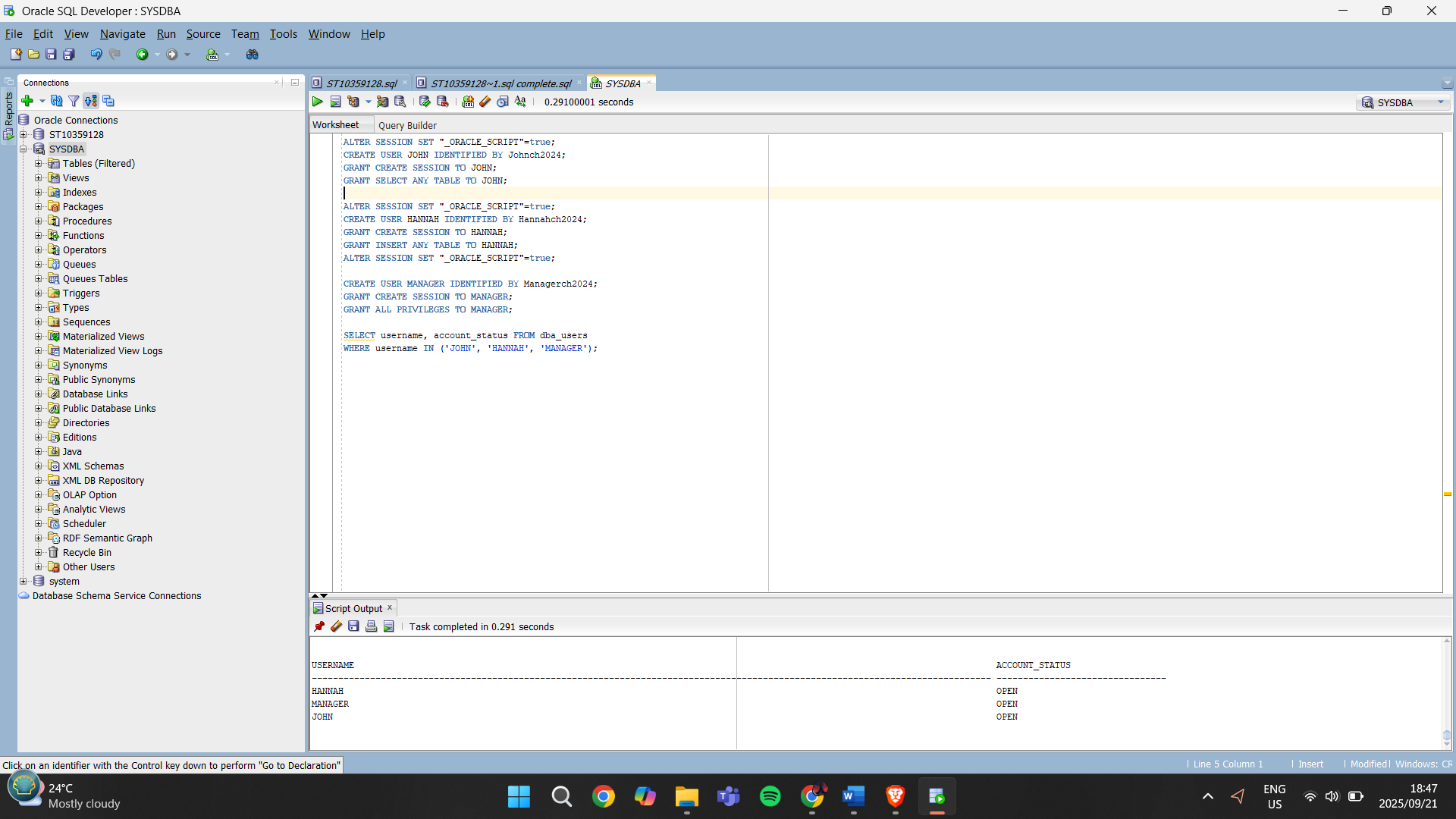
|  |
| --- |
| DRIVER |
| PK DriverID  FirstName  Surname  DriverCode  PhoneNumber  Address |

**QUESTION 2**



**QUESTION 3**

**Q.3.1.**

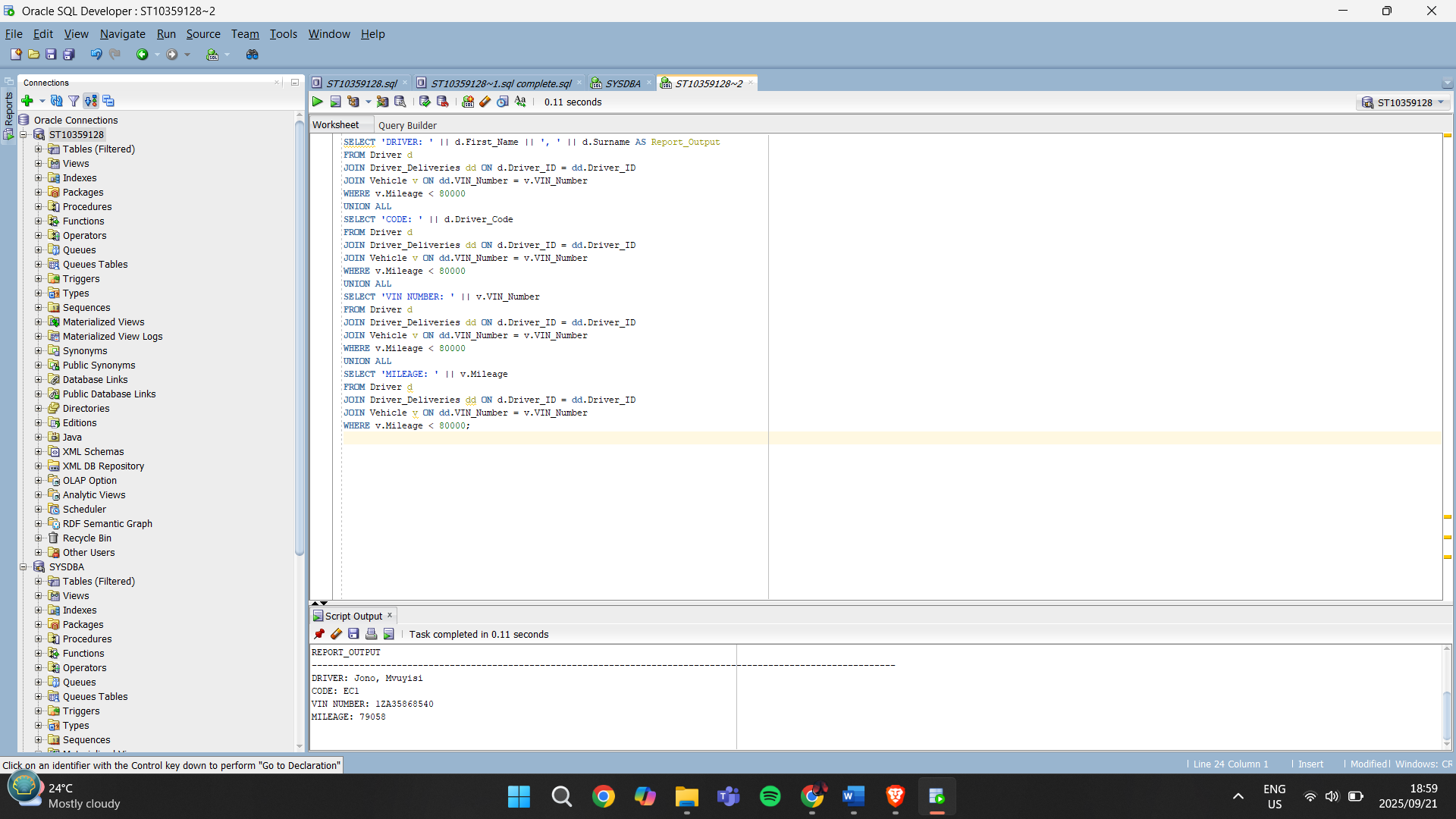


**Q.3.2.**

The idea of separating duties in a database is about security and accountability. Each user should only have the access they need to do their job. For example, John was only given SELECT rights so that he can read information but not accidentally change it. Hannah, on the other hand, can INSERT new records but cannot view everything in the system, which limits the risk of sensitive data exposure. By splitting these roles, the company reduces the chances of data leaks, errors, or abuse of privileges, and it also makes auditing easier because actions can be traced back to the correct user (Stair & Reynolds, 2019).

**QUESTION 4**

**Q.4.1.**

****

**Q.4.2.**

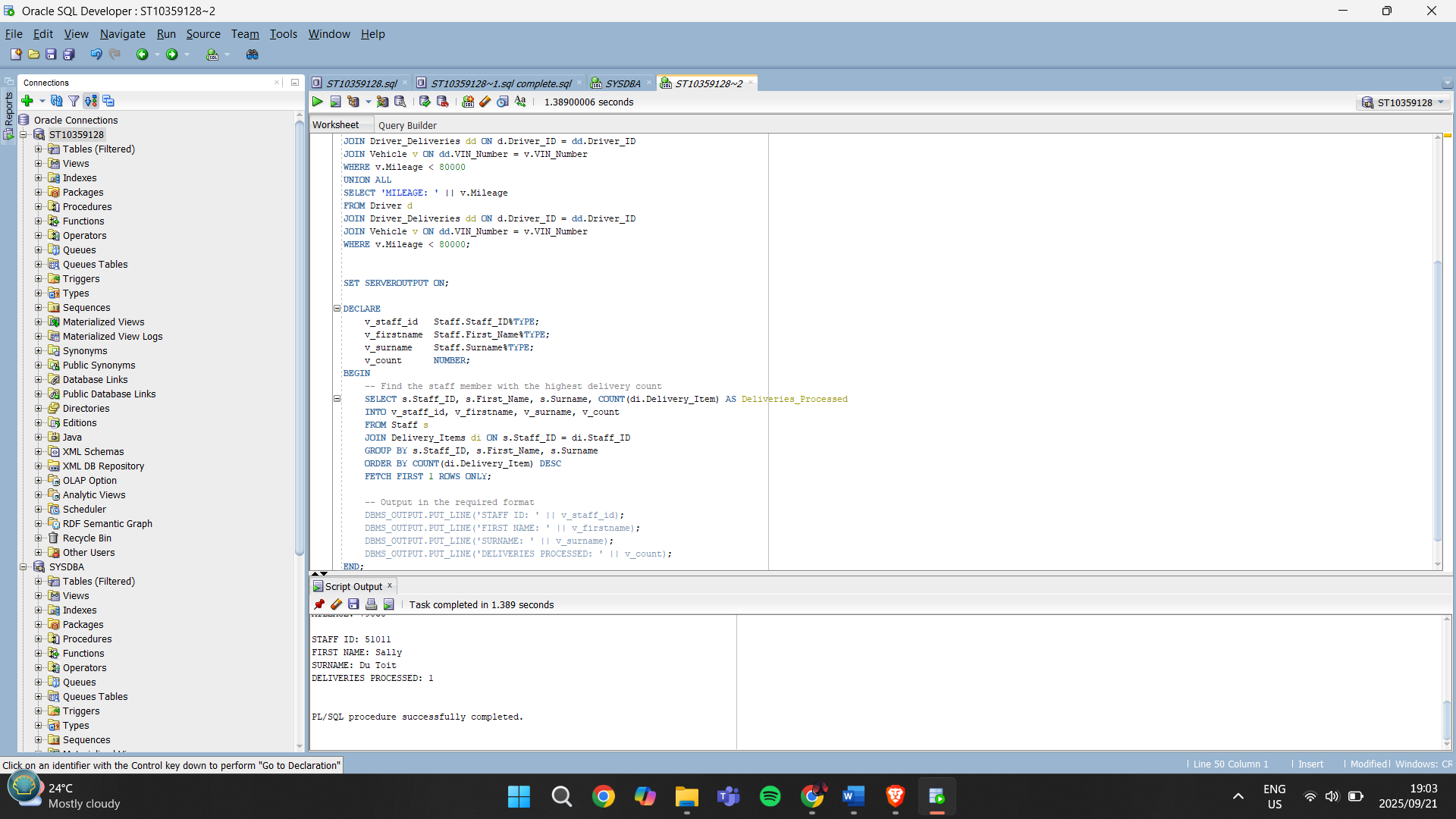
A flat file database is basically just a single table where all the data is stored together, usually in something like a CSV or Excel sheet. It’s simple to set up but it doesn’t handle complex relationships well. For example, with Cheetah Deliveries, if we stored customers, drivers, vehicles, and deliveries all in one flat file, there would be a lot of repeated data. This can lead to errors, duplication, and difficulty in managing large amounts of information (Coronel & Morris, 2015).

On the other hand, a relational model splits the data into multiple related tables that can be joined together through keys. This reduces duplication and improves accuracy because each entity has its own table, and relationships are maintained with primary and foreign keys (Elmasri & Navathe, 2016). For example, a driver only needs to be stored once in the Driver table, and their deliveries can be linked through a foreign key in the Deliveries table.

For Cheetah Deliveries, a relational model is clearly the better choice. The company deals with multiple entities like customers, drivers, vehicles, and delivery items, which are all related. By using a relational model, the business will be able to run reports more easily, ensure data accuracy, and avoid issues like redundancy that would happen with a flat file system. This makes it more efficient for managing operations and generating useful reports for decision-making (Connolly & Begg, 2015).

**QUESTION 5**

**Q.5.1.**

****

**Q.5.2.**

A PL/SQL block is made up of three main sections:

**Declaration Section**

This is where you create variables and data types that will be used in the block. For example, in Q.5.1, I declared variables like staff\_id, firstname, and count to temporarily store the results from the query. Without this, the block wouldn’t know where to keep the fetched data (Coronel & Morris, 2015).

**Execution Section**

This is the heart of the block where the SQL statements run. In my code, the execution section had the select statement to get the staff member with the most deliveries and the database output statements to display the report. This section always runs when the block is executed (Elmasri & Navathe, 2016).

**Exception Handling Section**

This section is used to catch and handle errors if something goes wrong. For example, if no staff records were found, Oracle could throw an exception. Exception handling makes the code more reliable (Connolly & Begg, 2015).

So overall, for my block I declared variables, executed the query and output, and could extend it further with exception handling.

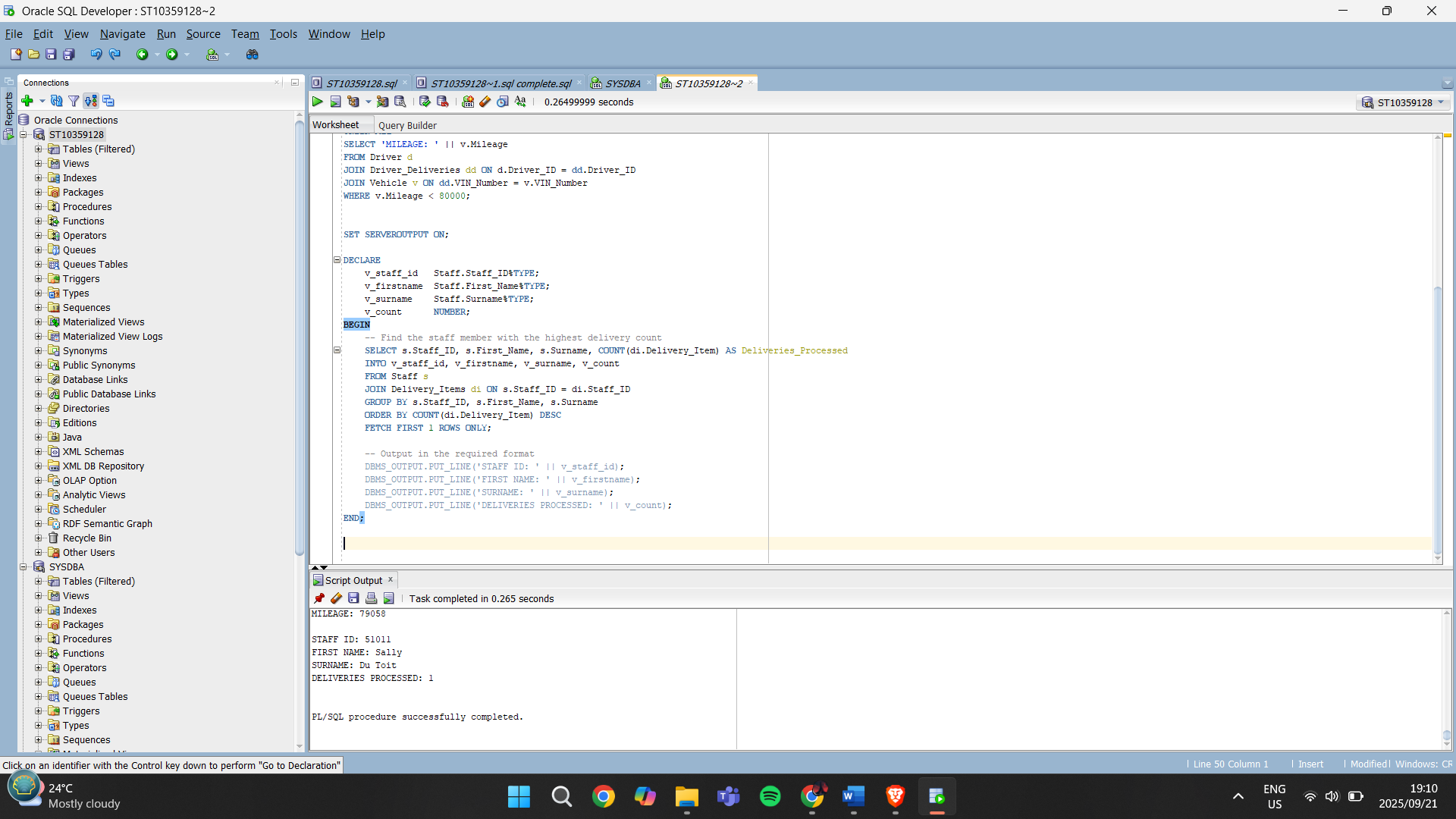
**Q.5.3.1.**

A View in SQL is a saved query that acts like a virtual table. Instead of storing data by itself, it shows results from one or more tables based on the SQL you define. For example, if management always needs to see which staff member has processed the most deliveries, we can create a view that joins Staff and Delivery tables, so the report can be pulled quickly without rewriting the query every time (Coronel & Morris, 2015).

The main benefit of a View is simplicity for the user. Managers don’t need to understand the technical SQL they can just run a select on the View as if it’s a normal table, and the results are ready. This also improves consistency, because everyone is using the same query rather than different versions (Connolly & Begg, 2015).

In short, a View makes it easier for non-technical users, like managers at Cheetah Deliveries, to generate reports without worrying about the joins and logic.

**Q.5.3.2.**



**QUESTION 6**

**Q.6.1.**

When working with PL/SQL, cursors are important because they let me control how results are handled. There are two types of cursors, and each is useful in different situations.

Implicit cursors are created automatically by Oracle whenever I run a SQL statement like **INSERT, UPDATE, DELETE, or a SELECT INTO**. They save me time because I don’t have to open or close them myself. I would use implicit cursor attributes when I only need to know whether a single row was affected. For example, at Cheetah Deliveries, if I update the mileage of a specific vehicle, I can quickly check if the update happened. This is efficient for one-row operations(Oracle, 2023; IBM, 2022).

Explicit cursors, on the other hand, are ones I create and control myself. I would use them when I expect multiple rows to be returned, and I want to process them one at a time. For example, if I want to generate a report that lists every driver and their details, I can open an explicit cursor and fetch each row in a loop. This gives me full control and is especially helpful in reports and batch processing for Cheetah Deliveries(Oracle, 2023; IBM, 2022).

So, implicit cursors are best for simple, single-row checks, while explicit cursors are better when I need to handle larger sets of data in a controlled way.

**CODE**

BEGIN

UPDATE Vehicle

SET Mileage = Mileage + 100

WHERE VIN\_Number = '12A35868540';

IF SQL%FOUND THEN

DBMS OUTPUT.PUT LINE('Vehicle mileage updated.');

ELSE

DBMS OUTPUT.PUT LINE('No vehicle found with that VIN.');

END IF;

END;

This code uses the SQL%FOUND attribute to check if a row was updated successfully(Oracle (2023), Silberschatz et al. (2020)).

DECLARE

CURSOR drv\_cur IS

SELECT Driver\_ID, Name, Surname FROM Driver;

v\_id Driver.Driver\_ID%TYPE;

v\_name Driver.Name%TYPE;

v\_sname Driver.Surname%TYPE;

BEGIN

OPEN drv\_cur;

LOOP

FETCH drv\_cur INTO v\_id, v\_name, v\_sname;

EXIT WHEN drv\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Driver: ' || v\_name || ' ' || v\_sname);

END LOOP;

CLOSE drv\_cur;

END;

This example shows how %NOTFOUND works with an explicit cursor to stop looping when there are no more drivers to fetch (Oracle (2023), Silberschatz et al. (2020)).

**Q.6.2.**

A sequence in Oracle is a database object that automatically generates unique numbers, usually used for primary keys. I would use a sequence in Cheetah Deliveries to generate Delivery\_IDs or Billing\_IDs. This ensures every new record gets a unique identifier without staff having to manually type it in, which reduces errors and duplication.

For example, every time a new delivery is added, the system can pull the next value from the sequence. This is especially useful when multiple staff members are working at the same time, because the database guarantees each ID is unique (Elmasri & Navathe, 2016).

CREATE SEQUENCE delivery\_seq

START WITH 1000

INCREMENT BY 1

NOCACHE

NOCYCLE;

INSERT INTO Delivery\_Items (Delivery\_Item, Staff\_ID)

VALUES ('NewPackage', delivery\_seq.NEXTVAL);

SELECT \* FROM Delivery\_Items;

The sequence delivery\_seq starts at 1000 and increases by 1 for each new record.

When inserting, delivery\_seq.NEXTVAL is used instead of typing an ID.

This means the first record will get ID 1000, the next will be 1001, and so on.

For Cheetah Deliveries, this makes delivery tracking easier and keeps the database consistent.

**REFERENCES:**

Connolly, T. & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management*. 6th ed. Harlow: Pearson Education.

Coronel, C. & Morris, S. (2015). *Database Systems: Design, Implementation, & Management*. 11th ed. Boston: Cengage Learning.

Coronel, C. & Morris, S. (2019). *Database Systems: Design, Implementation, & Management*. 13th ed. Boston: Cengage Learning.

Date, C.J. (2020). *An Introduction to Database Systems*. 8th ed. Boston: Pearson.

Elmasri, R. & Navathe, S.B. (2016). *Fundamentals of Database Systems*. 7th ed. Harlow: Pearson Education.

Oracle (2023). *Oracle Database SQL Language Reference*. Oracle. Available at: <https://docs.oracle.com/en/database/>

Stair, R. & Reynolds, G. (2019). *Principles of Information Systems*. 13th ed. Boston: Cengage Learning.

Silberschatz, A., Korth, H.F. & Sudarshan, S. (2020). Database System Concepts. 7th ed. New York: McGraw-Hill Education.