**Package**

1] Creating a package to calculate the gross pay, taxes and net pay amounts for a given employee

INSERT INTO employee\_payroll VALUES ('SW1011', TO\_DATE('2023-01-01', 'YYYY-MM-DD'), TIMESTAMP '2023-01-02 08:00:00', TIMESTAMP '2023-01-02 17:00:00', 10.00, 1);

CREATE OR REPLACE PACKAGE Employee\_Salary\_Package AS

PROCEDURE calculate\_gross\_salary(p\_emp\_id IN INT, p\_gross\_salary OUT NUMBER);

FUNCTION calculate\_total\_taxes(p\_gross\_salary IN NUMBER) RETURN NUMBER;

FUNCTION calculate\_net\_salary(p\_gross\_salary IN NUMBER, p\_taxes IN NUMBER) RETURN NUMBER;

END Employee\_Salary\_Package;

/

CREATE OR REPLACE PACKAGE BODY Employee\_Salary\_Package AS

PROCEDURE calculate\_gross\_salary(p\_emp\_id IN INT, p\_gross\_salary OUT NUMBER) AS

v\_gross\_salary NUMBER := 0;

v\_swipe\_total NUMBER;

v\_swipe\_id VARCHAR2(100);

BEGIN

FOR swipe\_record IN (SELECT swipe\_id FROM employee\_payroll WHERE emp\_id = p\_emp\_id) LOOP

v\_swipe\_id := swipe\_record.swipe\_id;

SELECT (EXTRACT(HOUR FROM (check\_out\_time - check\_in\_time)) +

EXTRACT(MINUTE FROM (check\_out\_time - check\_in\_time)) / 60) \* hourly\_pay

INTO v\_swipe\_total

FROM employee\_payroll

WHERE swipe\_id = v\_swipe\_id;

v\_gross\_salary := v\_gross\_salary + v\_swipe\_total;

END LOOP;

p\_gross\_salary := v\_gross\_salary;

END calculate\_gross\_salary;

FUNCTION calculate\_total\_taxes(p\_gross\_salary IN NUMBER) RETURN NUMBER AS

v\_total\_taxes NUMBER := 0;

BEGIN

IF p\_gross\_salary > 100 THEN

v\_total\_taxes := p\_gross\_salary \* 0.1; -- 10% tax

END IF;

RETURN v\_total\_taxes;

END calculate\_total\_taxes;

FUNCTION calculate\_net\_salary(p\_gross\_salary IN NUMBER, p\_taxes IN NUMBER) RETURN NUMBER AS

v\_net\_salary NUMBER;

BEGIN

v\_net\_salary := p\_gross\_salary - p\_taxes;

RETURN v\_net\_salary;

END calculate\_net\_salary;

END Employee\_Salary\_Package;

/

DECLARE

p\_emp\_id INT := 1;

v\_gross\_salary NUMBER;

v\_total\_taxes NUMBER;

v\_net\_salary NUMBER;

BEGIN

Employee\_Salary\_Package.calculate\_gross\_salary(p\_emp\_id, v\_gross\_salary);

v\_total\_taxes := Employee\_Salary\_Package.calculate\_total\_taxes(v\_gross\_salary);

v\_net\_salary := Employee\_Salary\_Package.calculate\_net\_salary(v\_gross\_salary, v\_total\_taxes);

DBMS\_OUTPUT.PUT\_LINE('Pay check details for Employee ID ' || p\_emp\_id || ':');

DBMS\_OUTPUT.PUT\_LINE('Gross salary: ' || TO\_CHAR(v\_gross\_salary, '99999.99'));

DBMS\_OUTPUT.PUT\_LINE('Total taxes: ' || TO\_CHAR(v\_total\_taxes, '99999.99'));

DBMS\_OUTPUT.PUT\_LINE('Net salary: ' || TO\_CHAR(v\_net\_salary, '99999.99'));

END;

/

**Procedures**

**1]** Creating a procedure to update the food quantity for animals with up to date vaccination status

CREATE OR REPLACE PROCEDURE update\_food\_quantity(changed\_quantity INT) AS

BEGIN

FOR rec IN (

SELECT a.animal\_id, n.food\_id, n.food\_quantity, a.a\_vaccine\_status

FROM animal\_info a

JOIN need ON a.animal\_id = need.animal\_id

JOIN nutrition n ON need.food\_id = n.food\_id

)

LOOP

IF rec.a\_vaccine\_status = 'Up to date' THEN

UPDATE nutrition

SET food\_quantity = food\_quantity - changed\_quantity

WHERE food\_id = rec.food\_id;

DBMS\_OUTPUT.PUT\_LINE('Food quantity updated for animal ' || rec.animal\_id || ': ' || (rec.food\_quantity - changed\_quantity));

END IF;

END LOOP;

END;

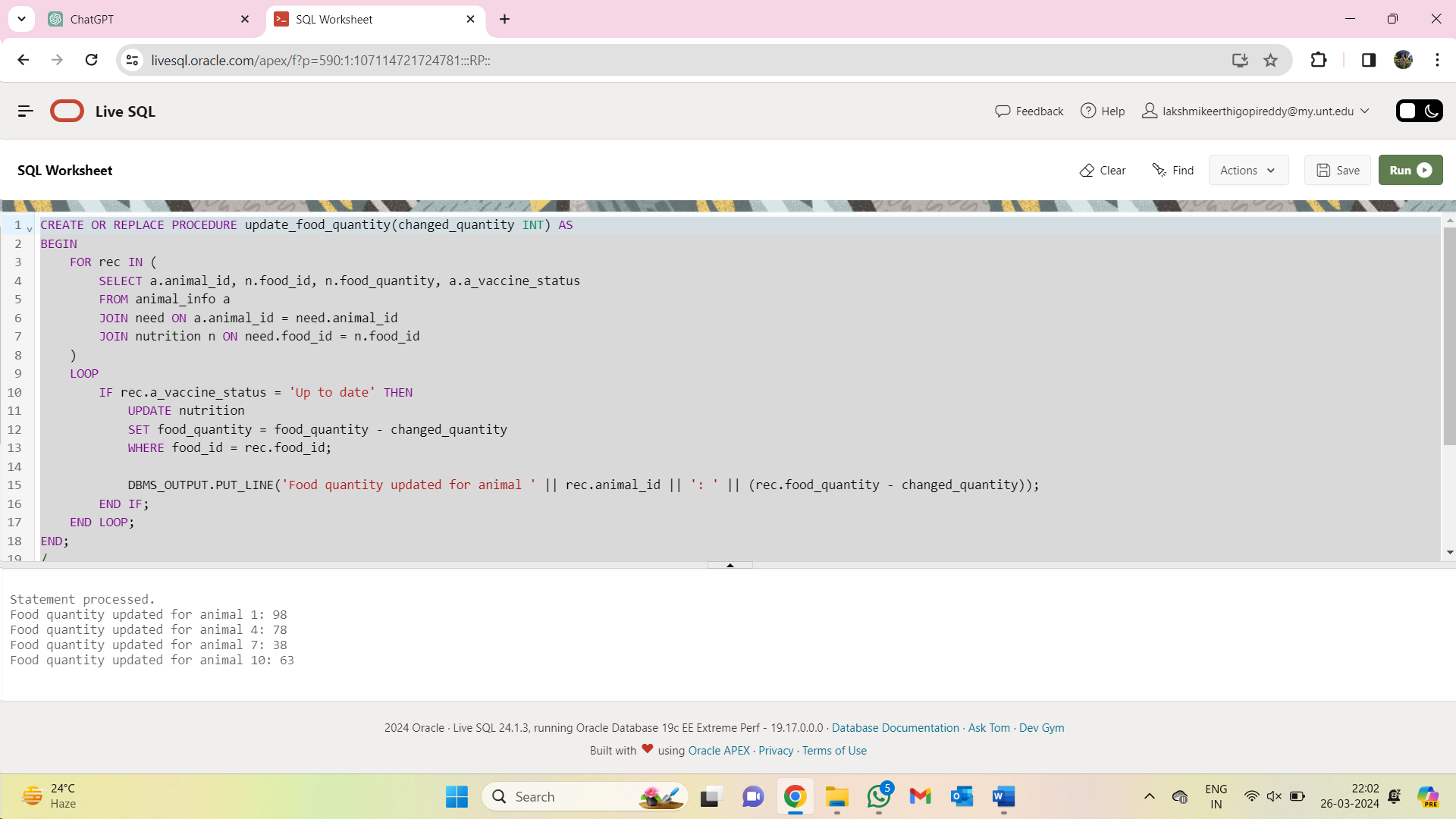
/

BEGIN

update\_food\_quantity(2); -- Call the procedure with the desired value for changed\_quantity

END;

/



**2]** Creating a procedure to get the details of Veterinarians with more than x years of work experience

CREATE OR REPLACE PROCEDURE GetVeterinariansByExperience(

p\_min\_experience IN INT,

p\_cursor OUT SYS\_REFCURSOR

) AS

BEGIN

-- Open a cursor to retrieve veterinarians with experience greater than or equal to the provided minimum experience

OPEN p\_cursor FOR

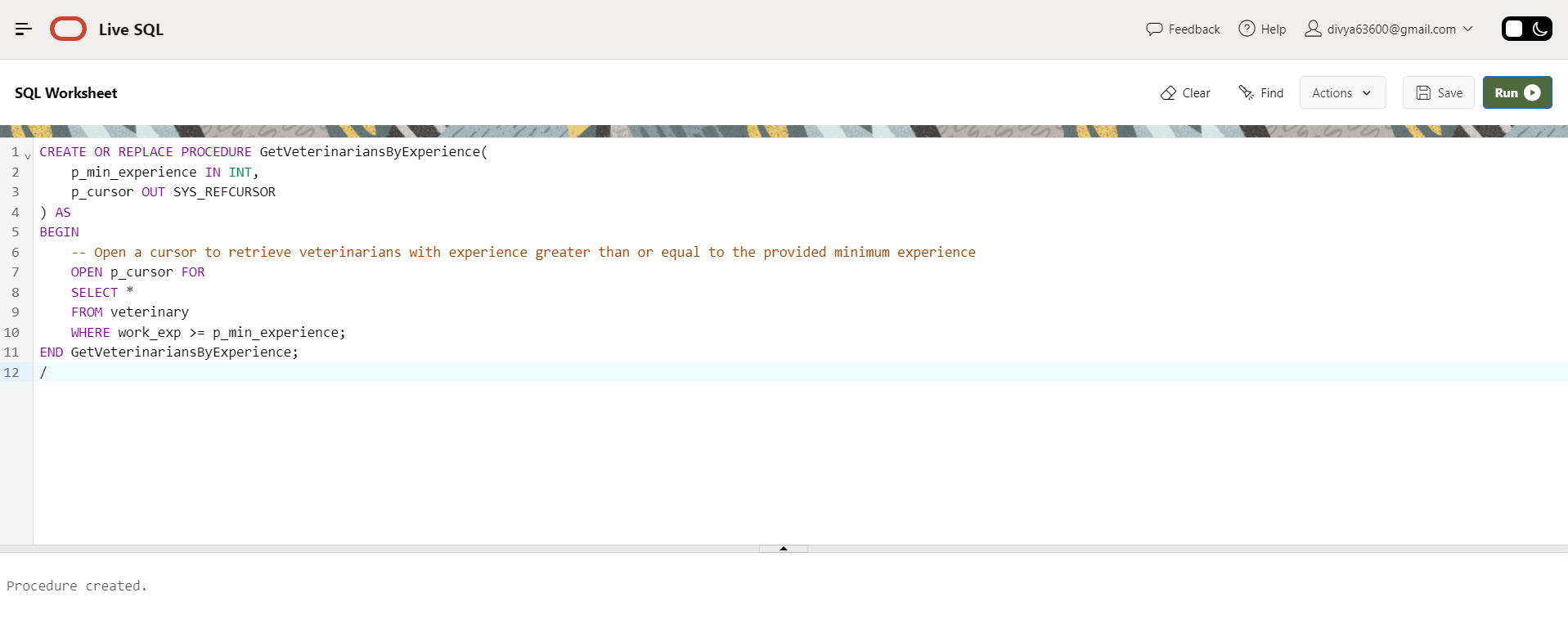
SELECT \*

FROM veterinary

WHERE work\_exp >= p\_min\_experience;

END GetVeterinariansByExperience;

/



DECLARE

vet\_cursor SYS\_REFCURSOR;

v\_vet\_id veterinary.vet\_id%TYPE;

v\_vet\_name veterinary.vet\_name%TYPE;

v\_vet\_contact veterinary.vet\_contact%TYPE;

v\_education veterinary.education%TYPE;

v\_work\_exp veterinary.work\_exp%TYPE;

BEGIN

-- Call the procedure to retrieve veterinarians with at least 10 years of experience

GetVeterinariansByExperience(10, vet\_cursor);

-- Fetch and display the results

LOOP

FETCH vet\_cursor INTO v\_vet\_id, v\_vet\_name, v\_vet\_contact, v\_education, v\_work\_exp;

EXIT WHEN vet\_cursor%NOTFOUND;

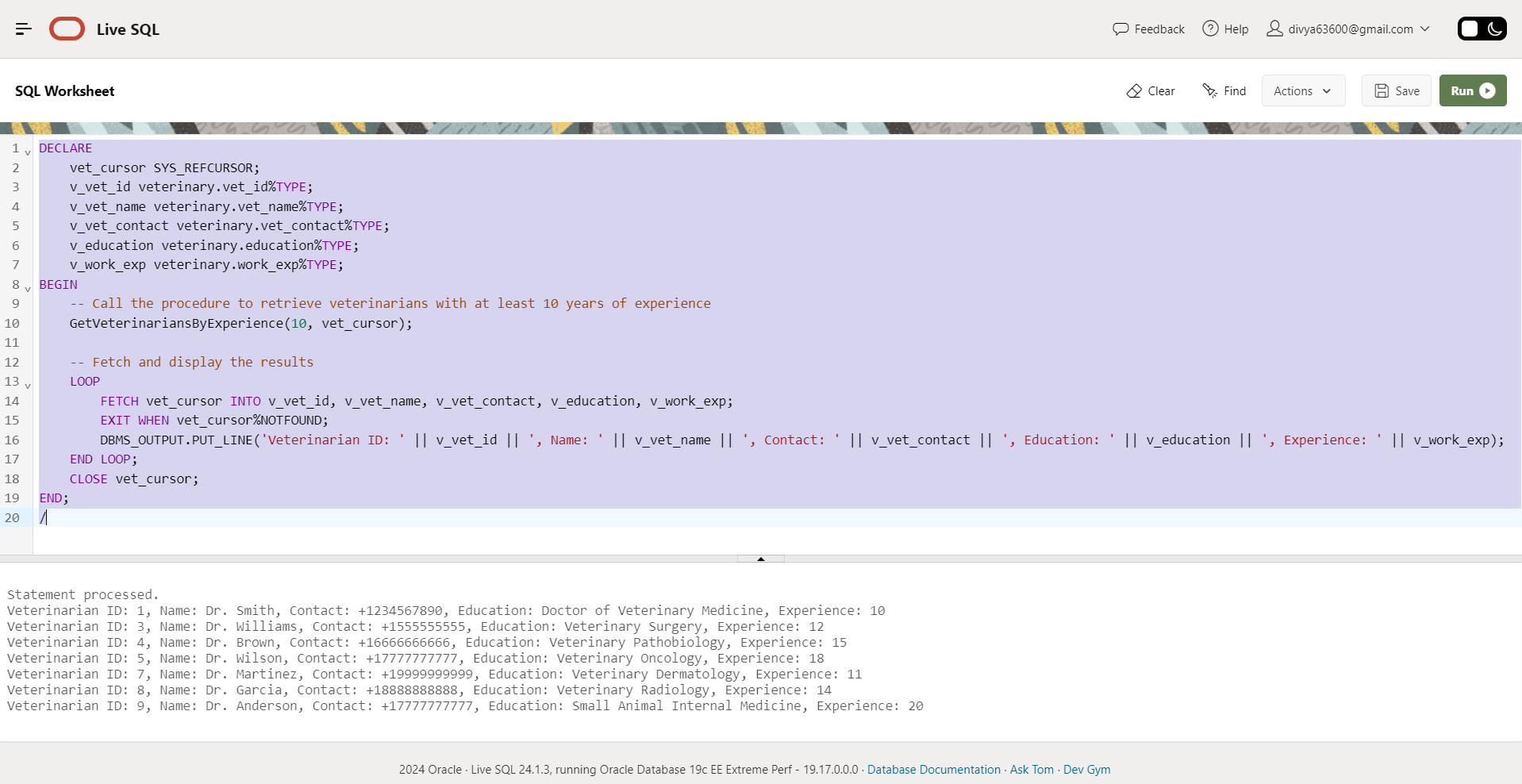
DBMS\_OUTPUT.PUT\_LINE('Veterinarian ID: ' || v\_vet\_id || ', Name: ' || v\_vet\_name || ', Contact: ' || v\_vet\_contact || ', Education: ' || v\_education || ', Experience: ' || v\_work\_exp);

END LOOP;

CLOSE vet\_cursor;

END;

/



3] Creating a procedure to display the vaccinations given on a particular date

CREATE OR REPLACE PROCEDURE DisplayVaccinationsForDate(

p\_vaccination\_date IN DATE

) AS

BEGIN

-- Display vaccinations for the given date

FOR rec IN (

SELECT vaccination\_name, vaccination\_date, vaccination\_dosage

FROM vaccination

WHERE vaccination\_date = p\_vaccination\_date

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Vaccination Name: ' || rec.vaccination\_name);

DBMS\_OUTPUT.PUT\_LINE('Vaccination Date: ' || TO\_CHAR(rec.vaccination\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Vaccination Dosage: ' || rec.vaccination\_dosage);

DBMS\_OUTPUT.PUT\_LINE('---------------------');

END LOOP;

END DisplayVaccinationsForDate;

/

DECLARE

-- Call the procedure to display vaccinations for a given date

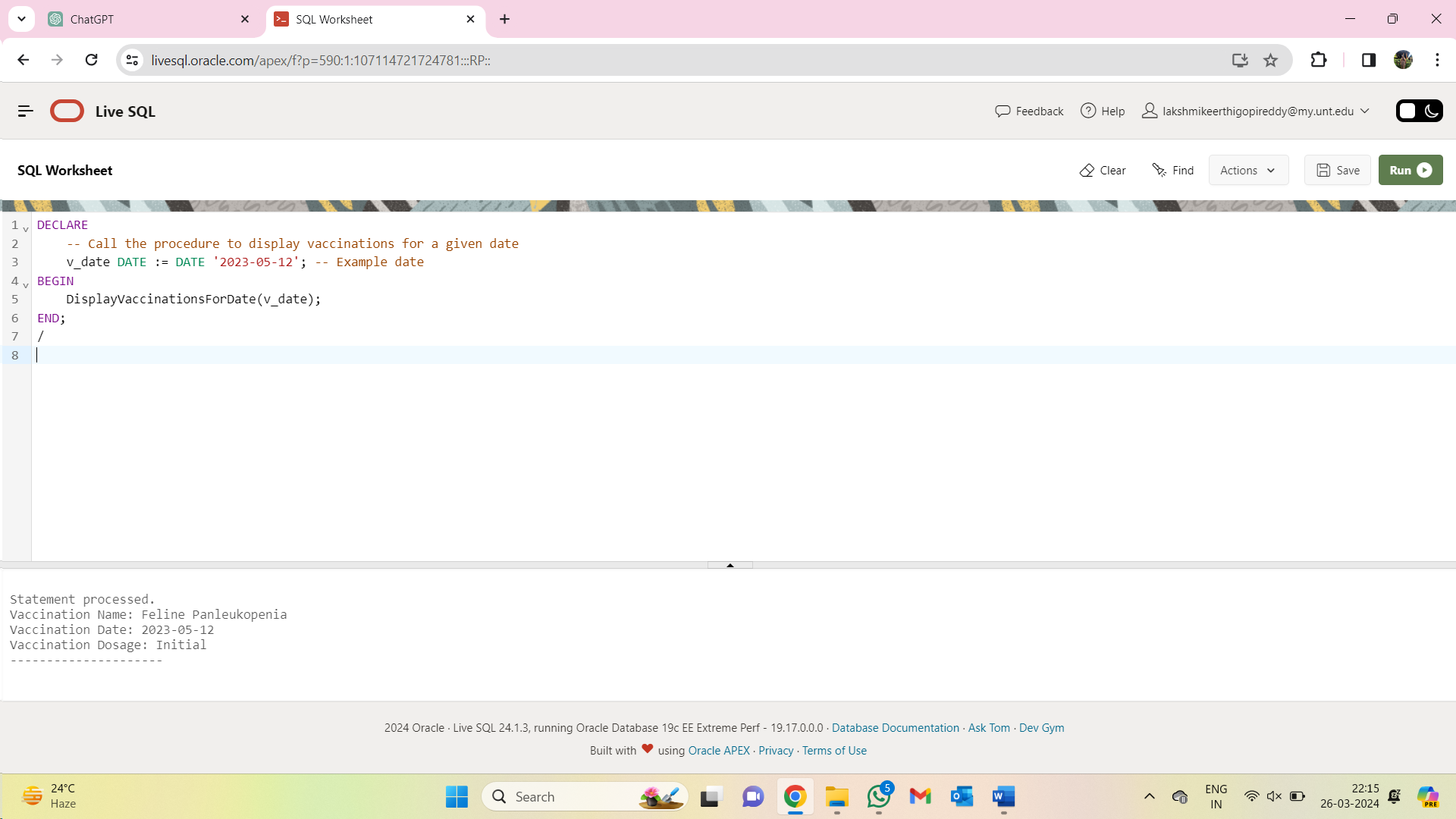
v\_date DATE := DATE '2023-05-12'; -- Example date

BEGIN

DisplayVaccinationsForDate(v\_date);

END;

/



4] Creating a procedure to get the vaccination name for a particular disease as per historical records.

CREATE OR REPLACE PROCEDURE GetVaccinationForDisease(

p\_disease IN VARCHAR2,

p\_vaccination\_name OUT VARCHAR2

) AS

BEGIN

-- Retrieve vaccination name based on the provided disease

SELECT v.vaccination\_name

INTO p\_vaccination\_name

FROM vaccination v

JOIN medical\_history m ON v.vaccination\_id = m.vaccination\_id

WHERE m.disease = p\_disease;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Handle case where no vaccination is found for the provided disease

DBMS\_OUTPUT.PUT\_LINE('No vaccination found for the provided disease.');

END GetVaccinationForDisease;

/

DECLARE

v\_vaccination\_name vaccination.vaccination\_name%TYPE;

v\_disease VARCHAR2(25) := ‘Infection’; -- Provide the disease name here

BEGIN

-- Call the procedure to retrieve vaccination name for the given disease

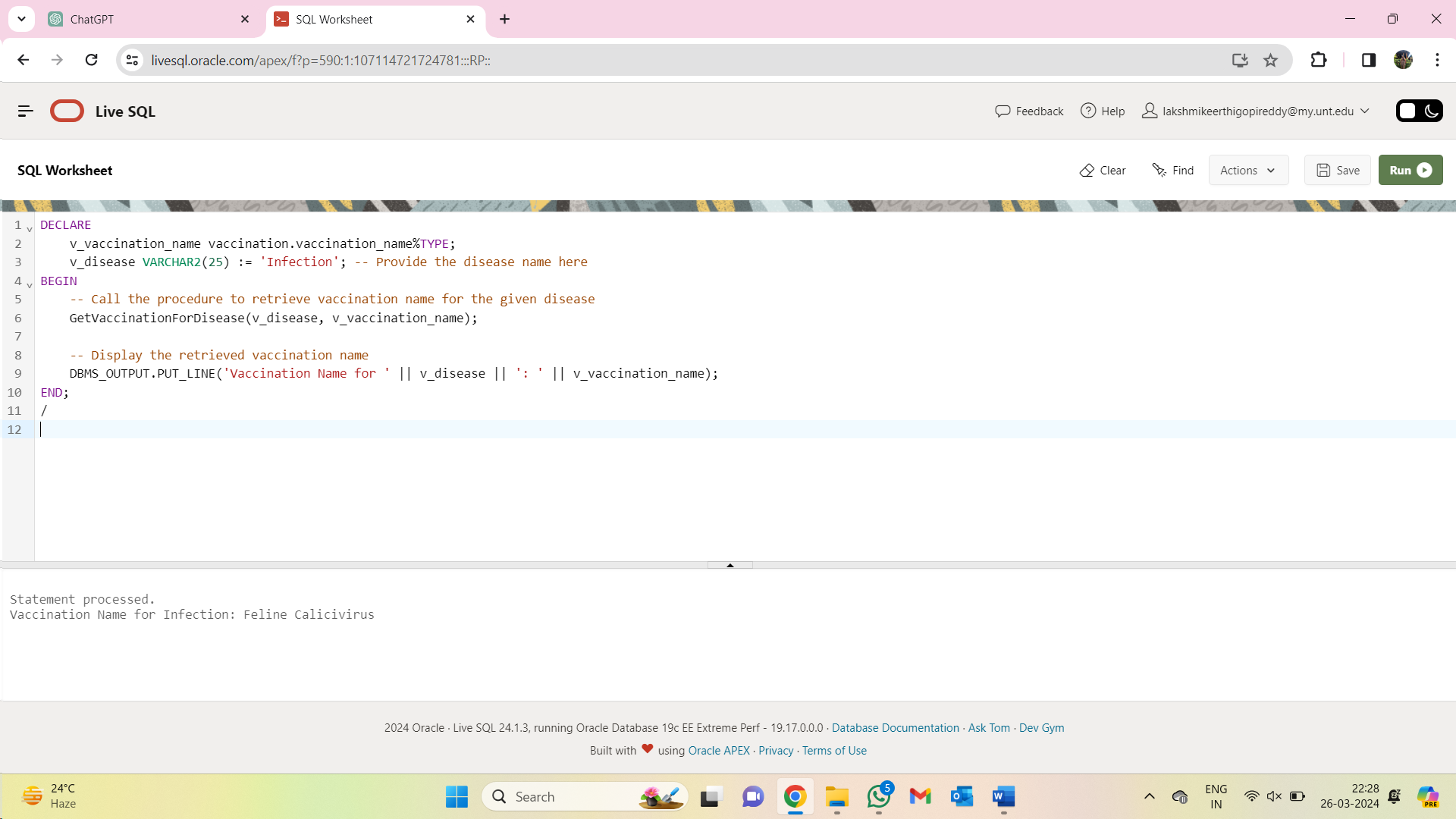
GetVaccinationForDisease(v\_disease, v\_vaccination\_name);

-- Display the retrieved vaccination name

DBMS\_OUTPUT.PUT\_LINE('Vaccination Name for ' || v\_disease || ': ' || v\_vaccination\_name);

END;

/



5] Creating a procedure to get the list of all the shelters with capacity more than the given number

CREATE OR REPLACE PROCEDURE GetSheltersWithCapacity(

p\_capacity\_threshold IN INT

) AS

BEGIN

-- Retrieve shelter names for shelters with capacity greater than the input value

FOR shelter\_rec IN (

SELECT sb.shelter\_name

FROM shelter\_branch sb

JOIN facilities f ON sb.shelter\_id = f.shelter\_id

WHERE f.f\_capacity > p\_capacity\_threshold

) LOOP

-- Display shelter names

DBMS\_OUTPUT.PUT\_LINE('Shelter Name: ' || shelter\_rec.shelter\_name);

END LOOP;

END GetSheltersWithCapacity;

/

DECLARE

v\_capacity\_threshold INT := 50; -- Provide the capacity threshold here

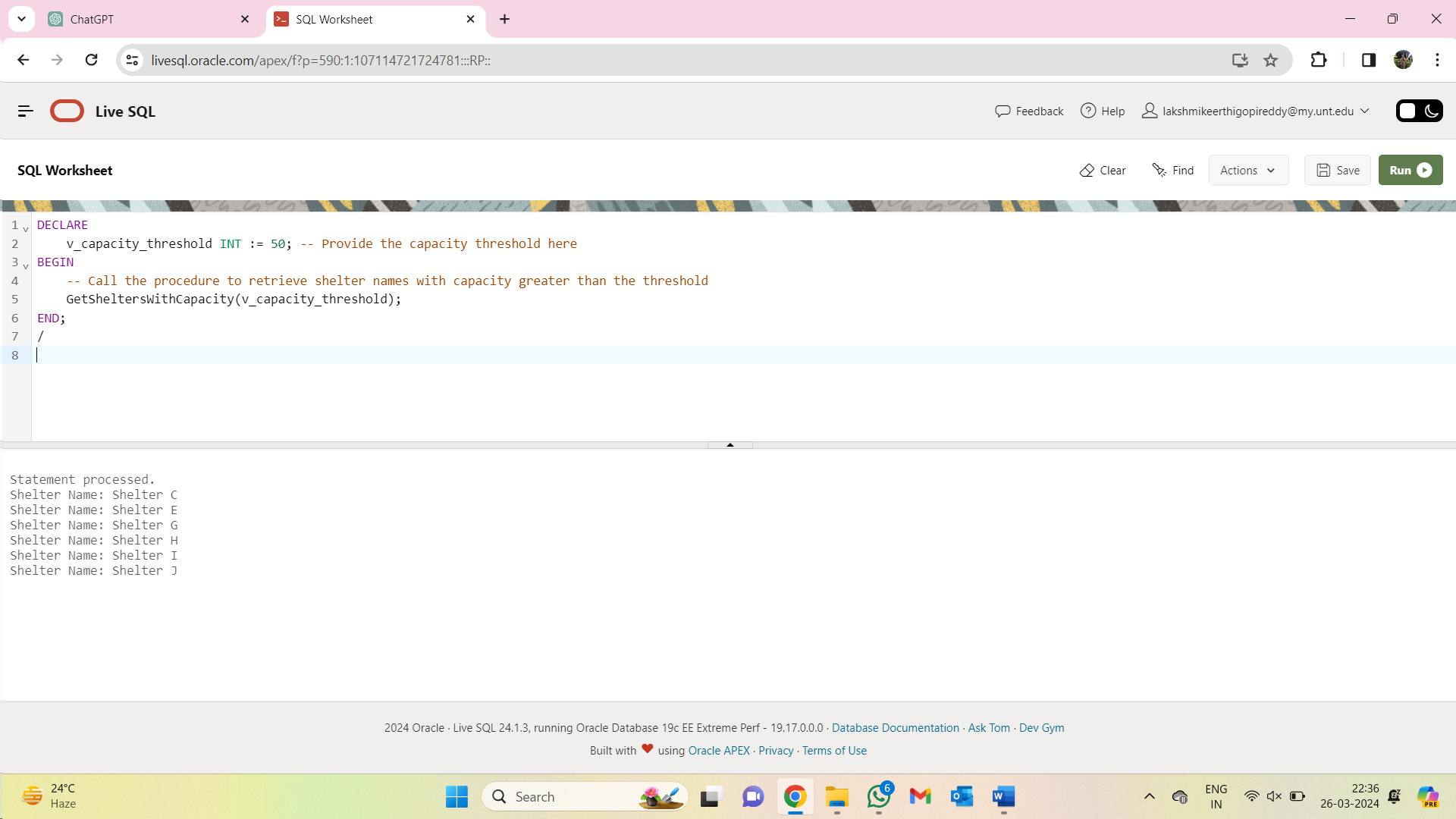
BEGIN

-- Call the procedure to retrieve shelter names with capacity greater than the threshold

GetSheltersWithCapacity(v\_capacity\_threshold);

END;

/



**Functions:**

1] Creating a function to get the count of the number of times an animal is transferred from one shelter to another.

CREATE OR REPLACE FUNCTION GetTransferCount(

p\_animal\_id INT

) RETURN INT AS

v\_transfer\_count INT;

BEGIN

-- Retrieve the number of transfers for the provided animal ID

SELECT COUNT(\*)

INTO v\_transfer\_count

FROM animal\_transfer\_records

WHERE animal\_id = p\_animal\_id;

-- Return the transfer count

RETURN v\_transfer\_count;

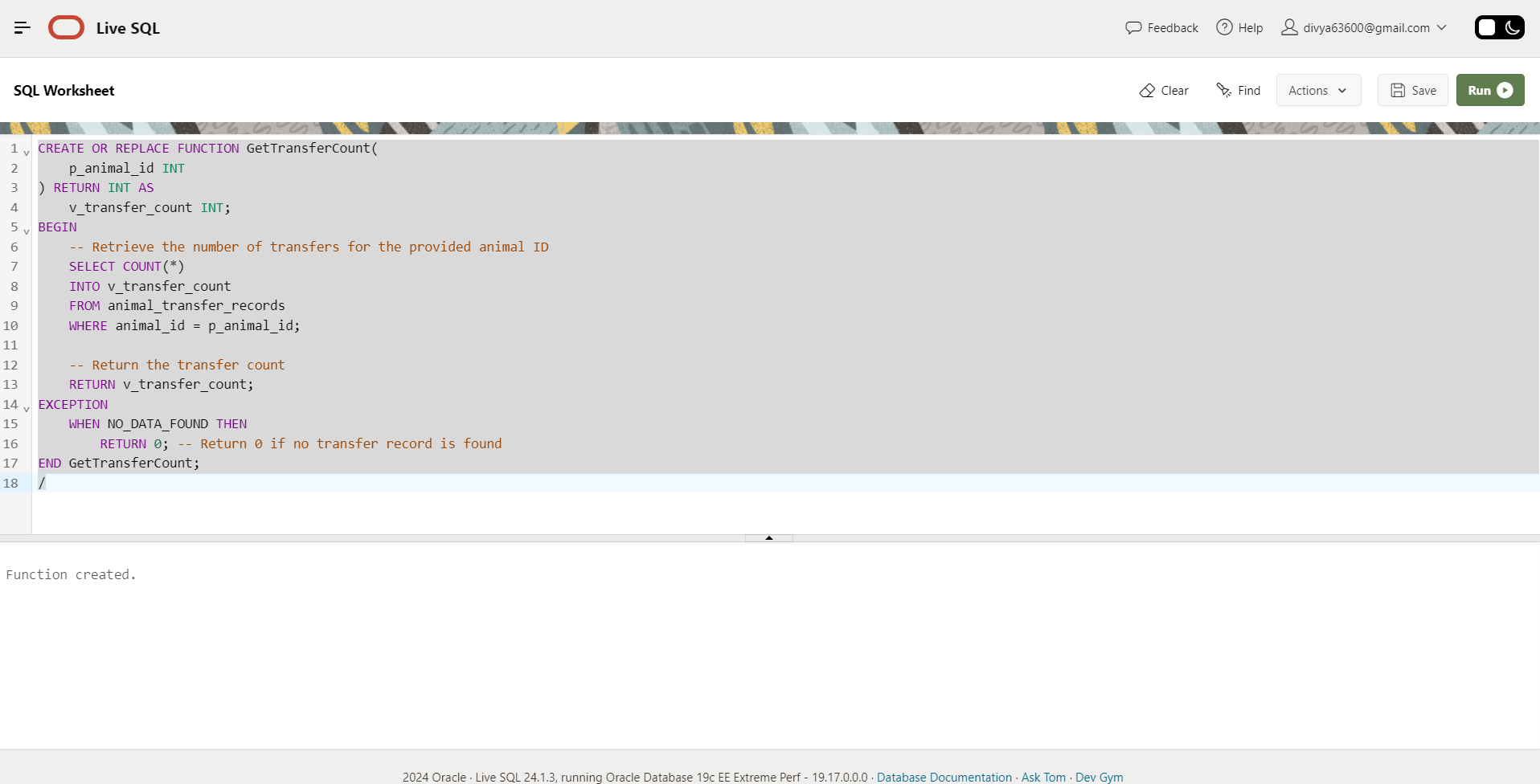
EXCEPTION

WHEN NO\_DATA\_FOUND THEN

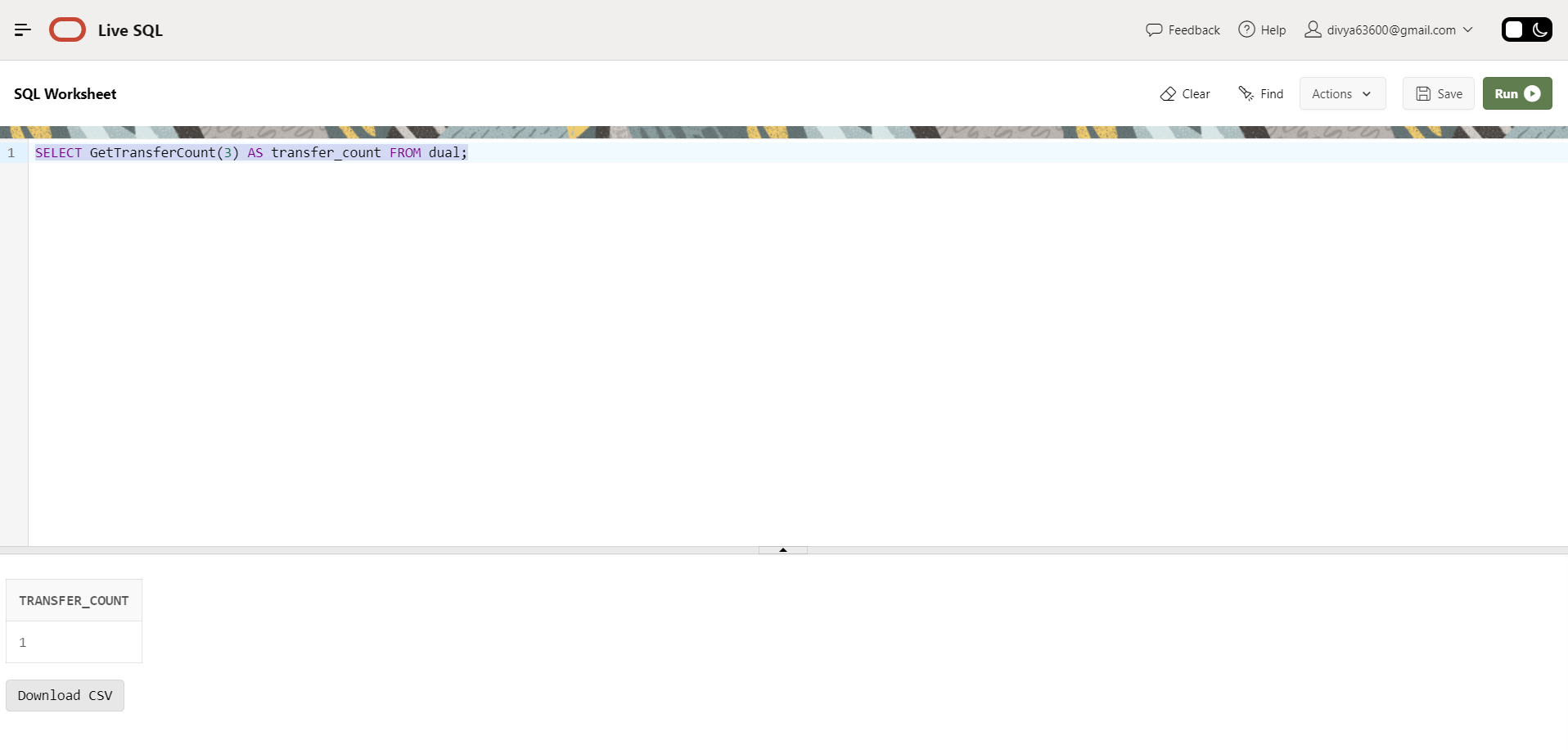
RETURN 0; -- Return 0 if no transfer record is found

END GetTransferCount;

/



SELECT GetTransferCount(3) AS transfer\_count FROM dual;



2] Creating a function to get the salary of a given employee

CREATE OR REPLACE FUNCTION GetEmployeeSalary(

p\_emp\_id INT

) RETURN NUMERIC AS

v\_salary NUMERIC;

BEGIN

-- Retrieve the salary of the employee with the provided ID

SELECT emp\_salary

INTO v\_salary

FROM employees\_staff\_details

WHERE emp\_id = p\_emp\_id;

-- Return the salary

RETURN v\_salary;

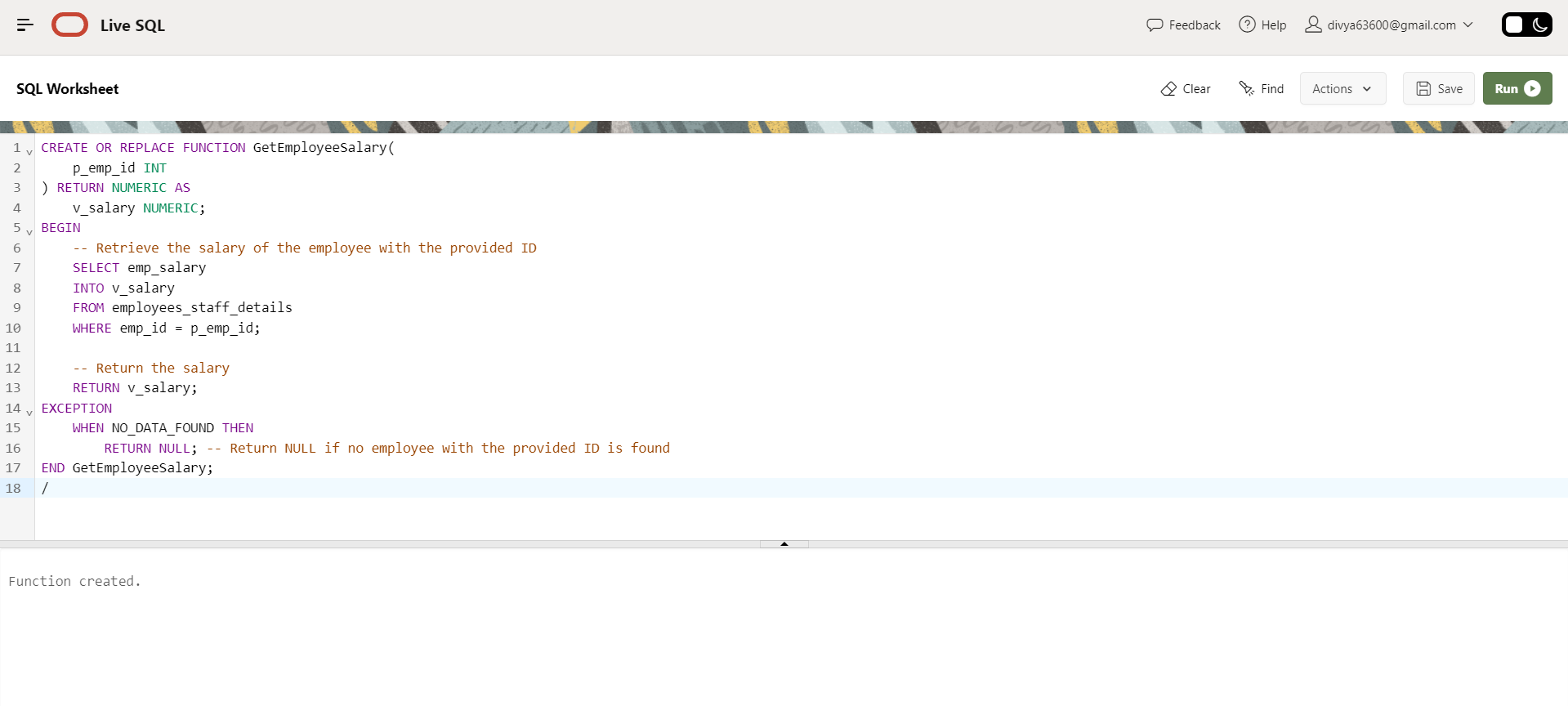
EXCEPTION

WHEN NO\_DATA\_FOUND THEN

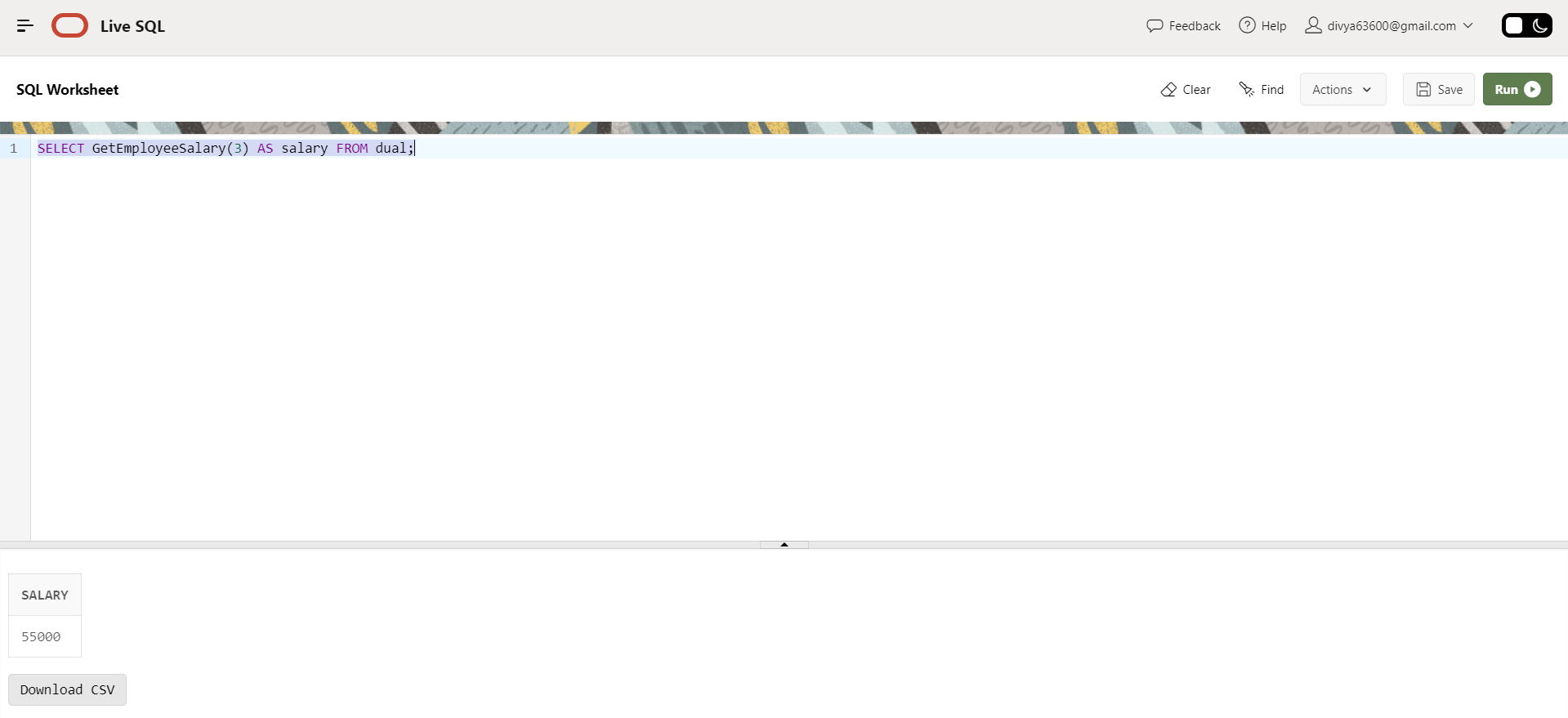
RETURN NULL; -- Return NULL if no employee with the provided ID is found

END GetEmployeeSalary;

/



SELECT GetEmployeeSalary(3) AS salary FROM dual;



3] Creating a function to check where outdoor space is available for a given shelter ID

CREATE OR REPLACE FUNCTION IsOutdoorAvailable(p\_shelter\_id IN INT) RETURN VARCHAR2 AS

outdoor\_status VARCHAR2(3);

BEGIN

SELECT CASE WHEN outdoor\_space = 'Y' THEN 'Yes' ELSE 'No' END

INTO outdoor\_status

FROM facilities

WHERE shelter\_id = p\_shelter\_id;

RETURN outdoor\_status;

END;

DECLARE

v\_outdoor\_status VARCHAR2(3);

v\_shelter\_id INT := 2; -- Provide the shelter ID for which you want to check outdoor availability

BEGIN

-- Call the function and retrieve the outdoor status

v\_outdoor\_status := IsOutdoorAvailable(v\_shelter\_id);

-- Display the result

IF v\_outdoor\_status = 'Yes' THEN

DBMS\_OUTPUT.PUT\_LINE('Outdoor space is available.');

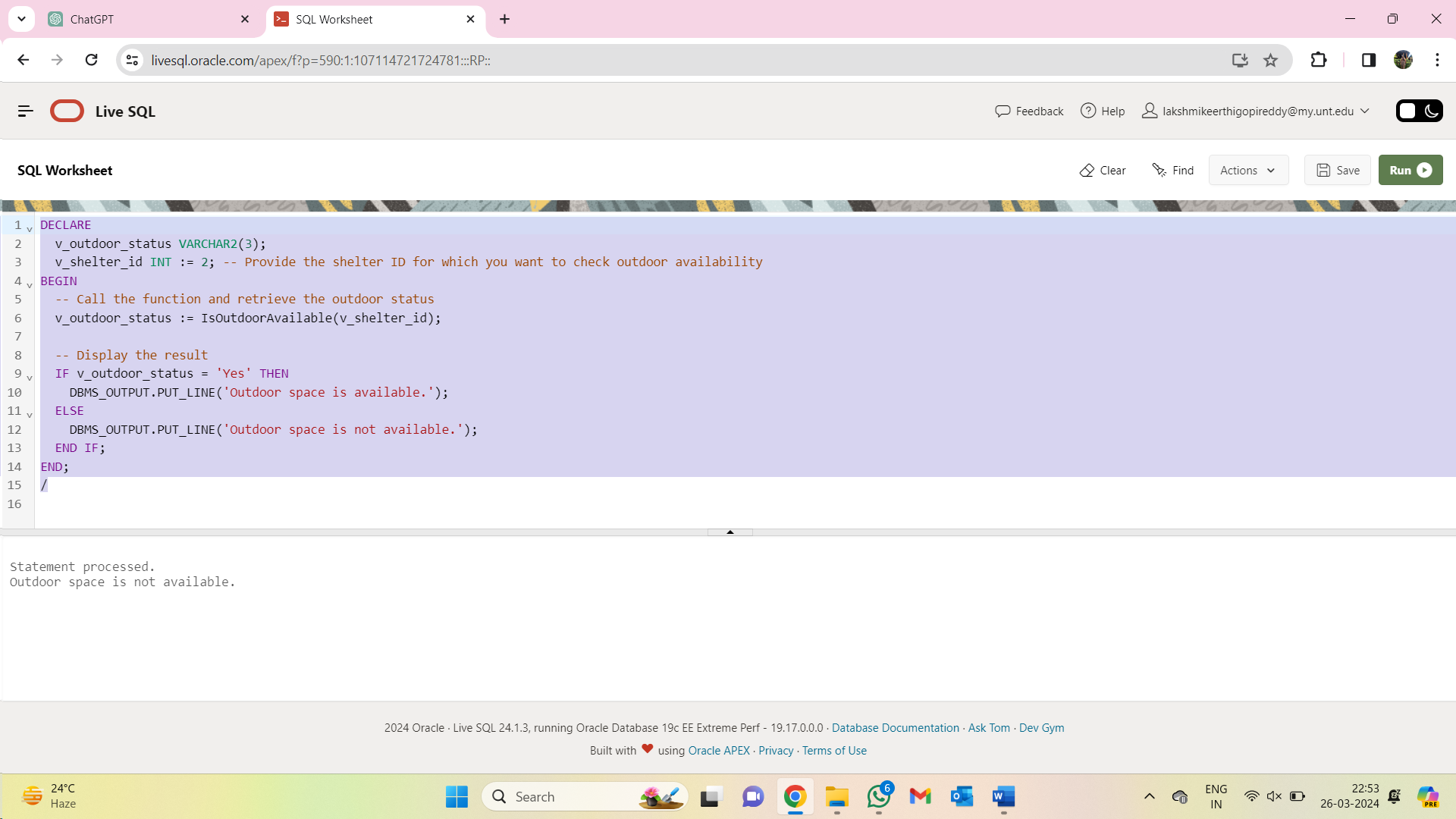
ELSE

DBMS\_OUTPUT.PUT\_LINE('Outdoor space is not available.');

END IF;

END;

/



4] Creating a function to check whether a new animal can be inserted into a shelter considering its capacity according to the facilities.

CREATE OR REPLACE FUNCTION IsSpaceAvailable(p\_shelter\_id IN INT) RETURN VARCHAR2 AS

v\_capacity INT;

v\_ani\_count INT;

BEGIN

-- Retrieve the capacity of the shelter's facility

SELECT f\_capacity

INTO v\_capacity

FROM facilities

WHERE shelter\_id = p\_shelter\_id;

-- Retrieve the current animal count for the specified shelter

SELECT COUNT(DISTINCT animal\_id)

INTO v\_ani\_count

FROM accommodate

WHERE shelter\_id = p\_shelter\_id;

-- Check if there is space available based on current animal count and facility capacity

IF v\_ani\_count < v\_capacity THEN

RETURN 'Space Available';

ELSE

RETURN 'No Space Available';

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'Shelter Not Found';

END;

DECLARE

v\_shelter\_id INT := 2; -- Provide the shelter ID you want to check

v\_space\_status VARCHAR2(20);

BEGIN

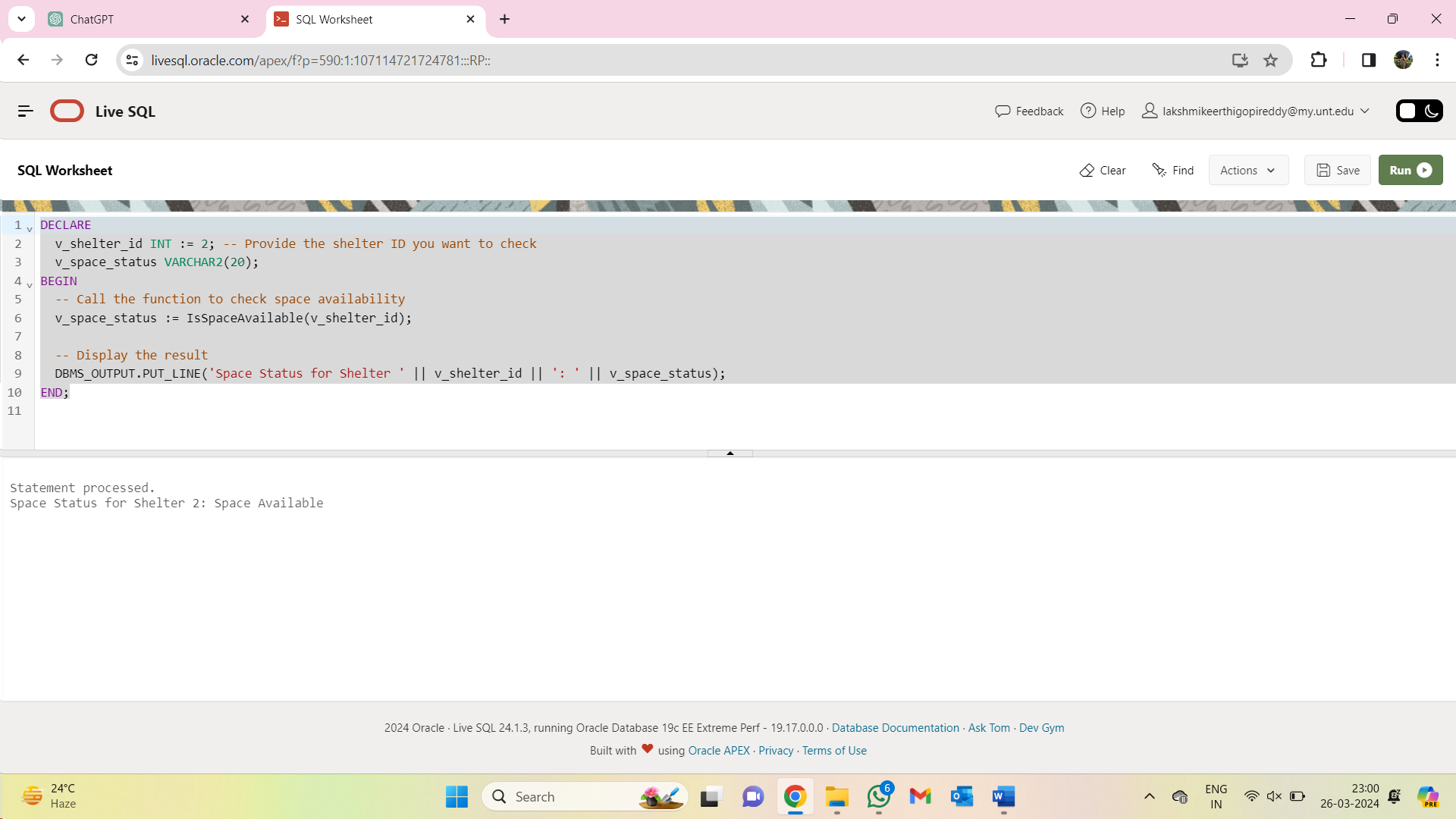
-- Call the function to check space availability

v\_space\_status := IsSpaceAvailable(v\_shelter\_id);

-- Display the result

DBMS\_OUTPUT.PUT\_LINE('Space Status for Shelter ' || v\_shelter\_id || ': ' || v\_space\_status);

END;



5] Creating a function to get the total funds from the sponsors for a particular shelter ID.

CREATE OR REPLACE FUNCTION GetFundsForShelter(

p\_shelter\_id IN INT

) RETURN NUMERIC AS

total\_funds NUMERIC := 0;

BEGIN

-- Calculate the total funds received by the shelter from all sponsors

SELECT SUM(amount) INTO total\_funds

FROM payment\_details

WHERE sponsor\_id IN (

SELECT sponsor\_id

FROM support

WHERE shelter\_id = p\_shelter\_id

);

RETURN total\_funds;

END;

/

DECLARE

v\_shelter\_id INT := 1; -- Provide the shelter ID for which you want to retrieve funds

v\_total\_funds NUMERIC;

BEGIN

-- Call the function to get total funds for the specified shelter

v\_total\_funds := GetFundsForShelter(v\_shelter\_id);

-- Display the total funds

DBMS\_OUTPUT.PUT\_LINE('Total funds for shelter ' || v\_shelter\_id || ': ' || v\_total\_funds);

END;

/

**Triggers**

1] Creating a trigger to calculate the adoption fee post discount (10% if 1000 or above, 5% if 750 or above) and update the value in table accordingly.

CREATE OR REPLACE TRIGGER CalculateDiscount

AFTER INSERT ON adoption\_details

FOR EACH ROW

DECLARE

v\_payment\_amount adoption\_details.ad\_fee%TYPE;

BEGIN

-- Retrieve the payment amount for the new adoption

v\_payment\_amount := :NEW.ad\_fee;

-- Check if the payment amount is eligible for a discount

IF v\_payment\_amount >= 1000 THEN

-- Apply a 10% discount for payments of 1000 or more

:NEW.ad\_fee := v\_payment\_amount \* 0.9;

ELSIF v\_payment\_amount >= 750 THEN

-- Apply a 5% discount for payments between 750 and 999

:NEW.ad\_fee := v\_payment\_amount \* 0.95;

END IF;

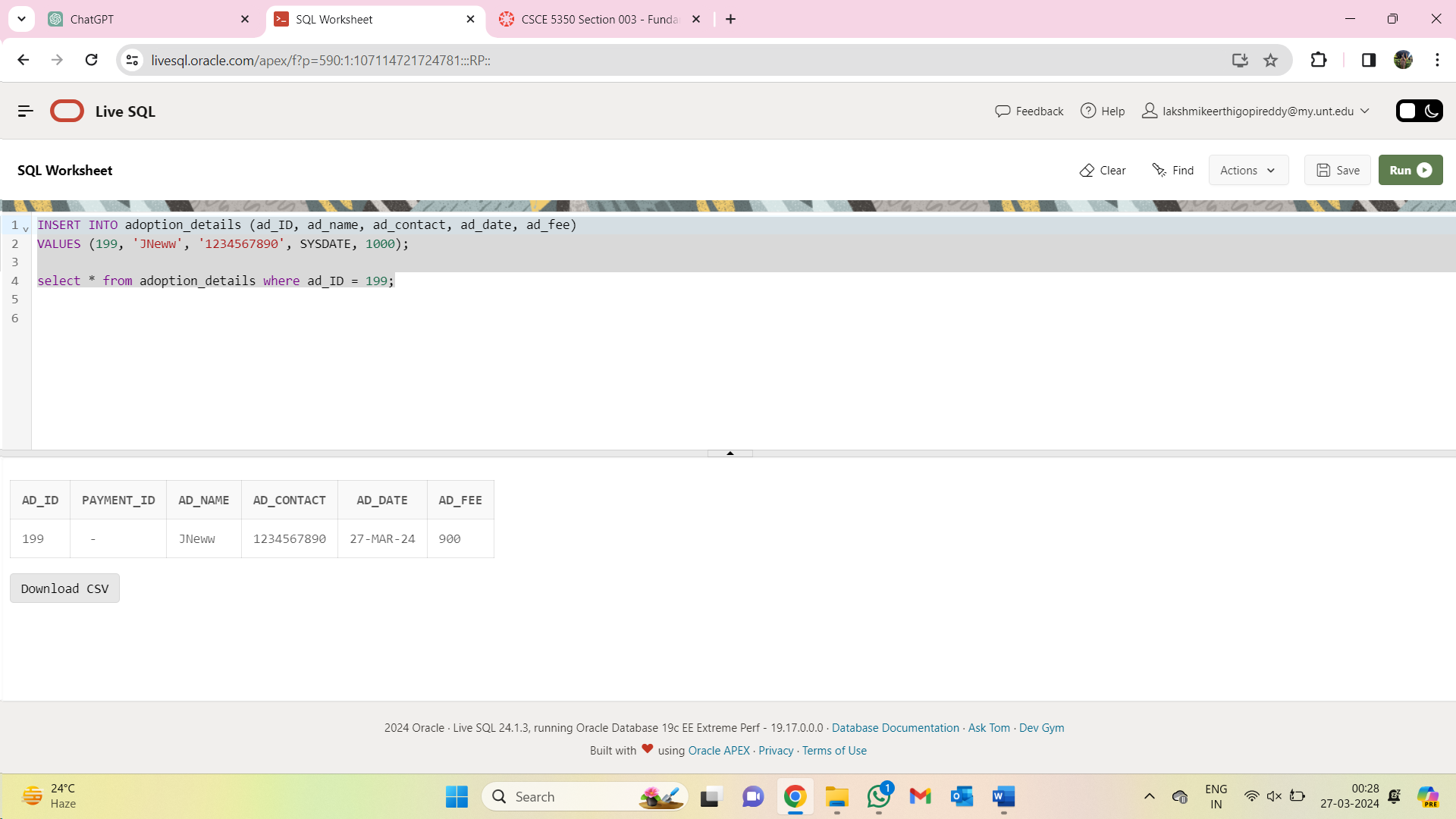
END;

/

INSERT INTO adoption\_details (ad\_ID, ad\_name, ad\_contact, ad\_date, ad\_fee)

VALUES (199, 'JNeww', '1234567890', SYSDATE, 1000);

select \* from adoption\_details where ad\_ID = 199;



2] Creating a trigger to ensure that the reason for surrendering an animal does not exceed more than 20 characters.

CREATE OR REPLACE TRIGGER CheckSurrenderReason

BEFORE INSERT OR UPDATE ON surrender\_details

FOR EACH ROW

DECLARE

v\_reason\_length INT;

BEGIN

-- Check the length of the surrender reason

v\_reason\_length := LENGTH(:NEW.s\_reason);

-- Enforce a maximum length for the surrender reason

IF v\_reason\_length > 20 THEN

-- If the reason length exceeds the limit, raise an error

RAISE\_APPLICATION\_ERROR(-20001, 'Surrender reason cannot exceed 20 characters.');

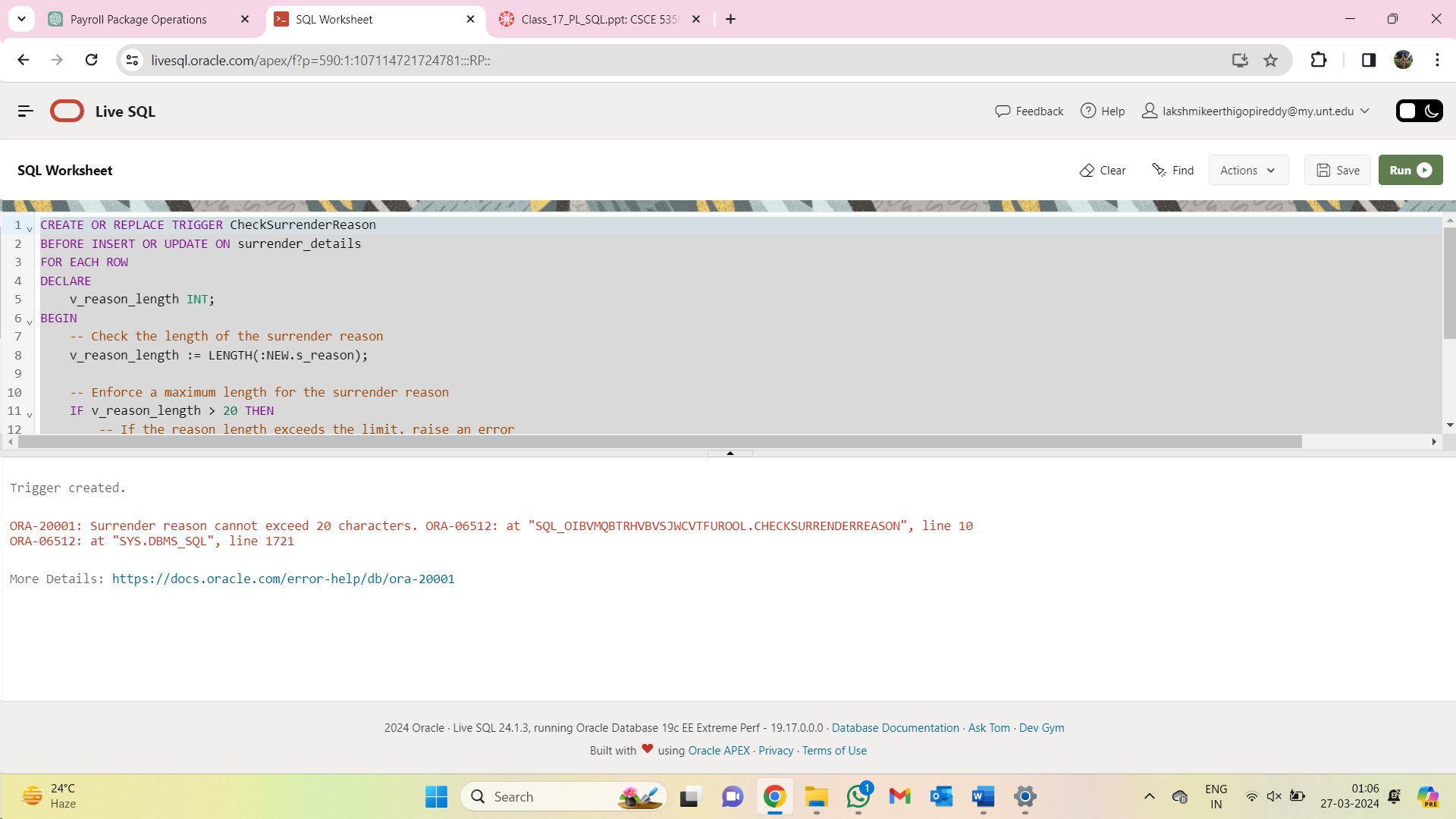
END IF;

END;

/

INSERT INTO surrender\_details (s\_id, s\_date, s\_name, s\_reason)

VALUES (1011, SYSDATE, 'John Doe', 'The reason for surrendering the pet is due to unforeseen circumstances beyond my control, including financial constraints, health issues, and changes in living arrangements');



3] Creating a trigger to ensure that the date of acquisition of an animal is always after the date of birth of the animal.

CREATE OR REPLACE TRIGGER CheckDOB

BEFORE INSERT ON animal\_info

FOR EACH ROW

BEGIN

IF :NEW.a\_dob > :NEW.date\_of\_acquisition THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Date of birth cannot be greater than date of acquisition.');

END IF;

END;

/

INSERT INTO animal\_info (animal\_id, a\_type, a\_breed, a\_dob, date\_of\_acquisition, a\_vaccine\_status)

VALUES (133, 'Dog', 'Labrador Retriever', DATE '2023-03-15', DATE '2023-01-01', 'Up to date');

