**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.
  + **Code**

CREATE TABLE Customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

date\_of\_birth DATE,

balance NUMBER(10,2),

IsVIP CHAR(1),

email VARCHAR2(100)

);

CREATE TABLE Loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER REFERENCES Customers(customer\_id),

due\_date DATE,

interest\_rate NUMBER(5,2)

);

INSERT INTO Customers VALUES (1, 'Ravi Sharma', TO\_DATE('1950-06-20', 'YYYY-MM-DD'), 12000, 'N', 'ravi@example.com');

INSERT INTO Customers VALUES (2, 'Sneha Mehra', TO\_DATE('1985-09-15', 'YYYY-MM-DD'), 9500, 'N', 'sneha@example.com');

INSERT INTO Customers VALUES (3, 'Sunita Ghosh', TO\_DATE('1949-01-10', 'YYYY-MM-DD'), 20000, 'N', 'sunita@example.com');

INSERT INTO Customers VALUES (4, 'Arjun Verma', TO\_DATE('2000-03-05', 'YYYY-MM-DD'), 8000, 'N', 'arjun@example.com');

INSERT INTO Customers VALUES (5, 'Priya Kapoor', TO\_DATE('1961-11-30', 'YYYY-MM-DD'), 11000, 'N', 'priya@example.com');

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10, 8.5);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 40, 9.0);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5, 7.5);

INSERT INTO Loans VALUES (104, 4, SYSDATE + 25, 8.0);

INSERT INTO Loans VALUES (105, 5, SYSDATE + 60, 9.5);

COMMIT;

BEGIN

FOR rec IN (

SELECT c.customer\_id, l.loan\_id, l.interest\_rate,

FLOOR(MONTHS\_BETWEEN(SYSDATE, c.date\_of\_birth) / 12) AS age

FROM Customers c

JOIN Loans l ON c.customer\_id = l.customer\_id

) LOOP

IF rec.age > 60 THEN

UPDATE Loans

SET interest\_rate = interest\_rate - 1

WHERE loan\_id = rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE('1% interest discount applied for customer ID ' || rec.customer\_id);

END IF;

END LOOP;

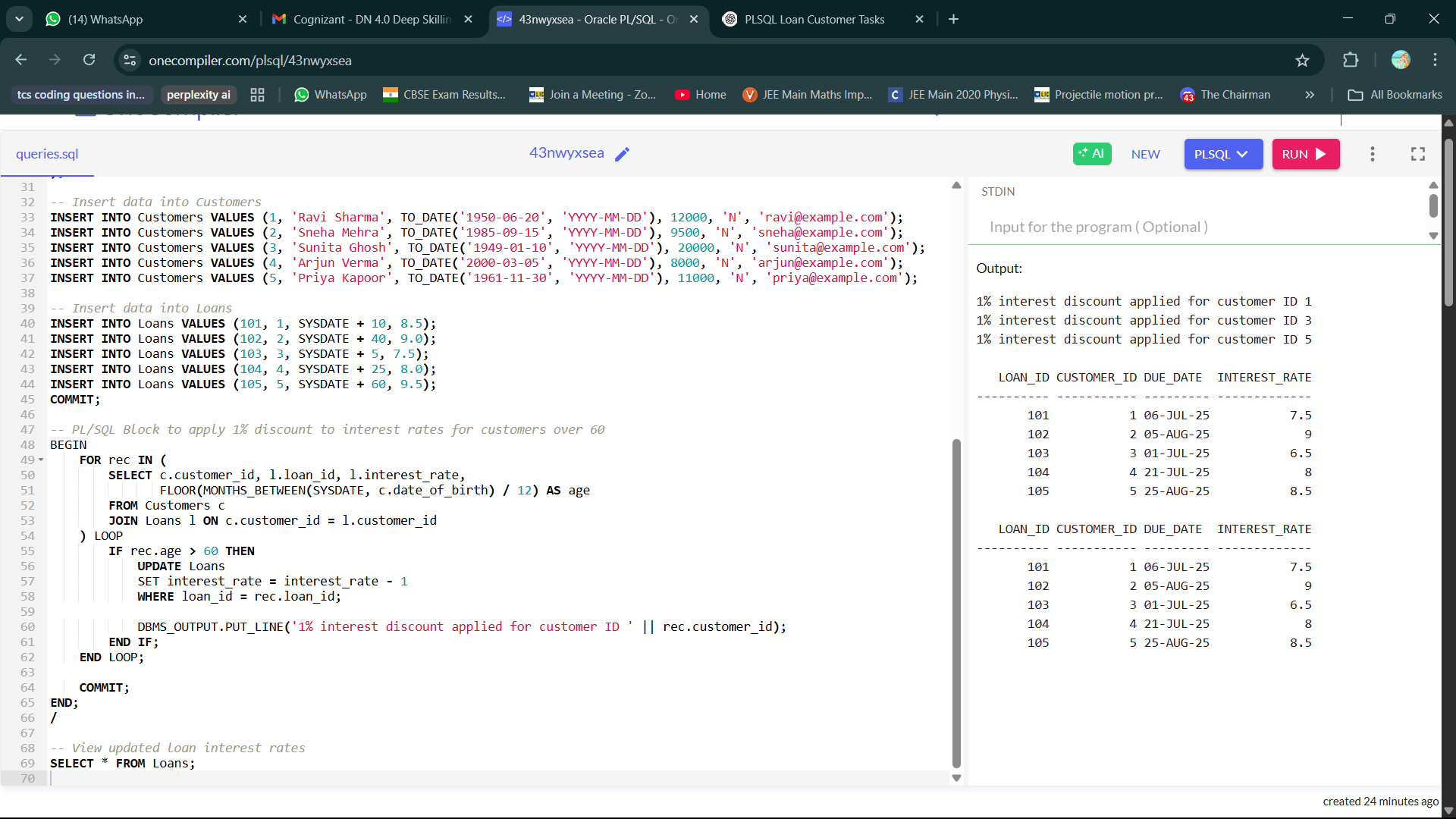
COMMIT;

END;

/

SELECT \* FROM Loans;

* + **Output**



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.
  + **Code**

BEGIN

FOR rec IN (SELECT customer\_id, balance FROM Customers) LOOP

IF rec.balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.customer\_id || ' promoted to VIP.');

END IF;

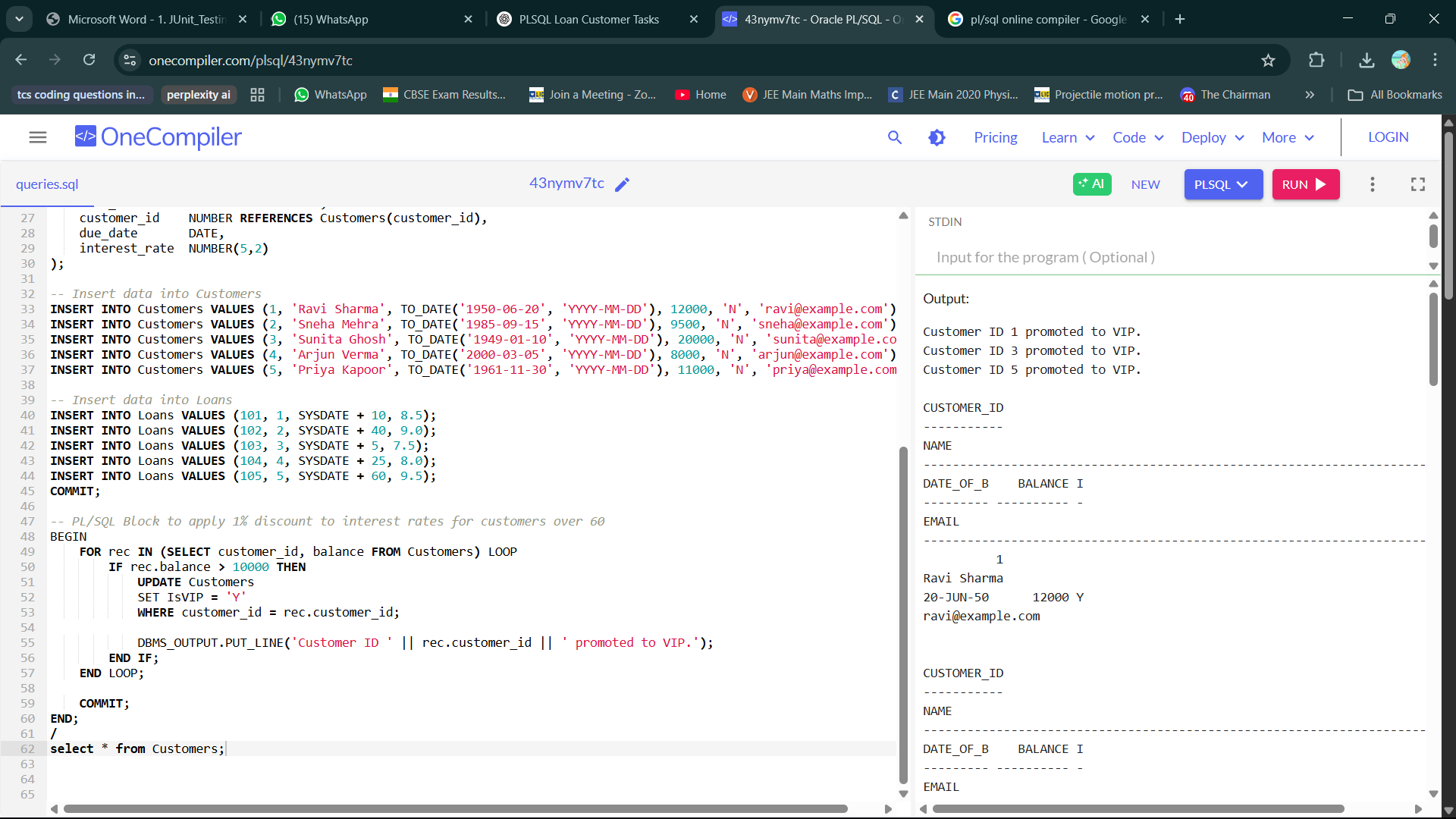
END LOOP;

COMMIT;

END;

/

* + **Output**



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.
* **Code:**

BEGIN

FOR rec IN (

SELECT c.name, c.email, l.loan\_id, l.due\_date

FROM Customers c

JOIN Loans l ON c.customer\_id = l.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.loan\_id ||

' for ' || rec.name || ' is due on ' ||

TO\_CHAR(rec.due\_date, 'DD-MON-YYYY') ||

'. Email: ' || rec.email);

END LOOP;

END;

/

* **Output**

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.
  + **Code**

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20), -- 'SAVINGS' or 'CURRENT'

balance NUMBER(12, 2)

);

INSERT INTO Accounts VALUES (101, 1, 'SAVINGS', 10000);

INSERT INTO Accounts VALUES (102, 2, 'CURRENT', 15000);

INSERT INTO Accounts VALUES (103, 3, 'SAVINGS', 20000);

INSERT INTO Accounts VALUES (104, 4, 'SAVINGS', 30000);

INSERT INTO Accounts VALUES (105, 5, 'CURRENT', 5000);

COMMIT;

CREATE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET balance = balance + (balance \* 0.01)

WHERE UPPER(account\_type) = 'SAVINGS';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

END;

/

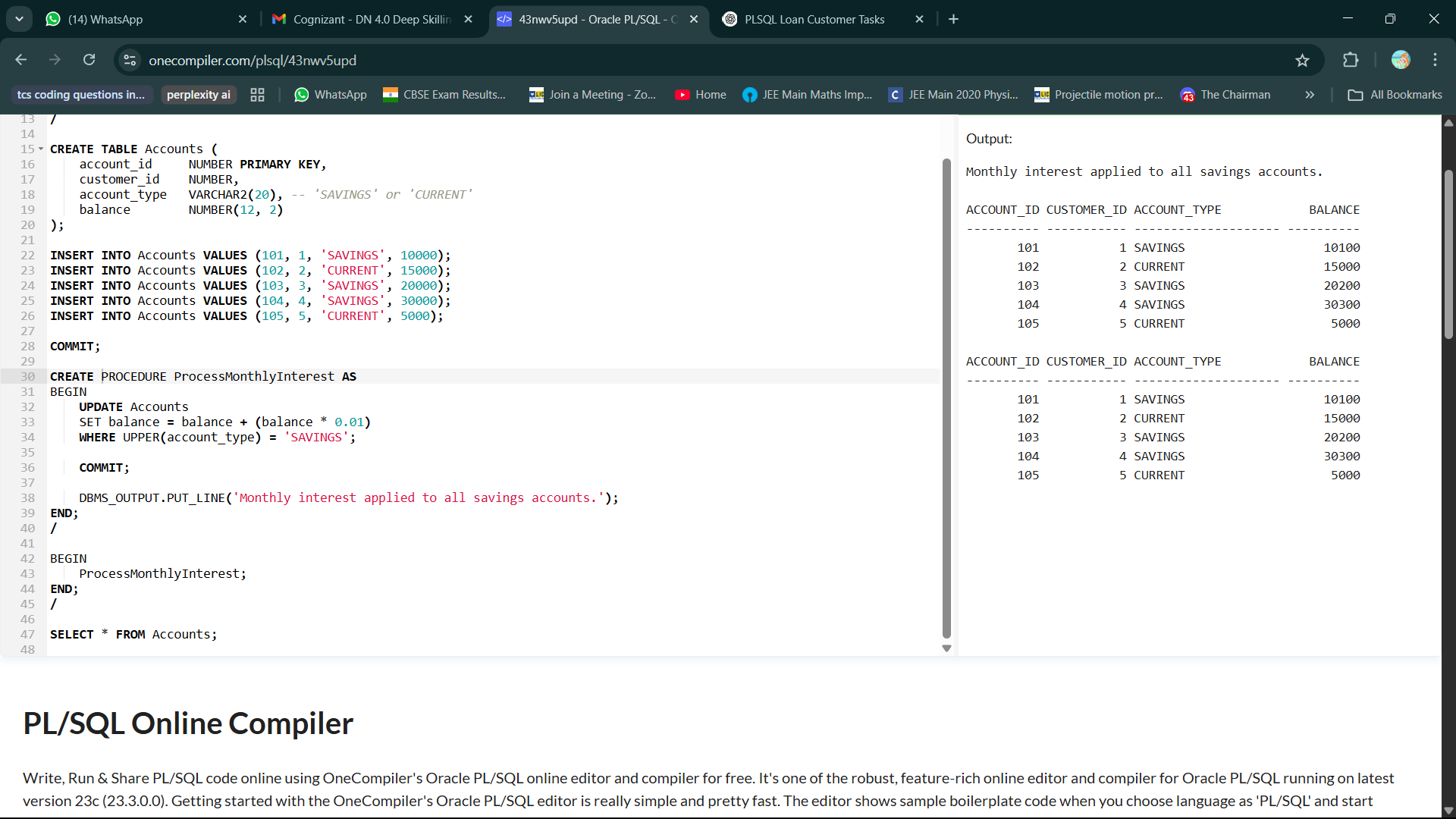
BEGIN

ProcessMonthlyInterest;

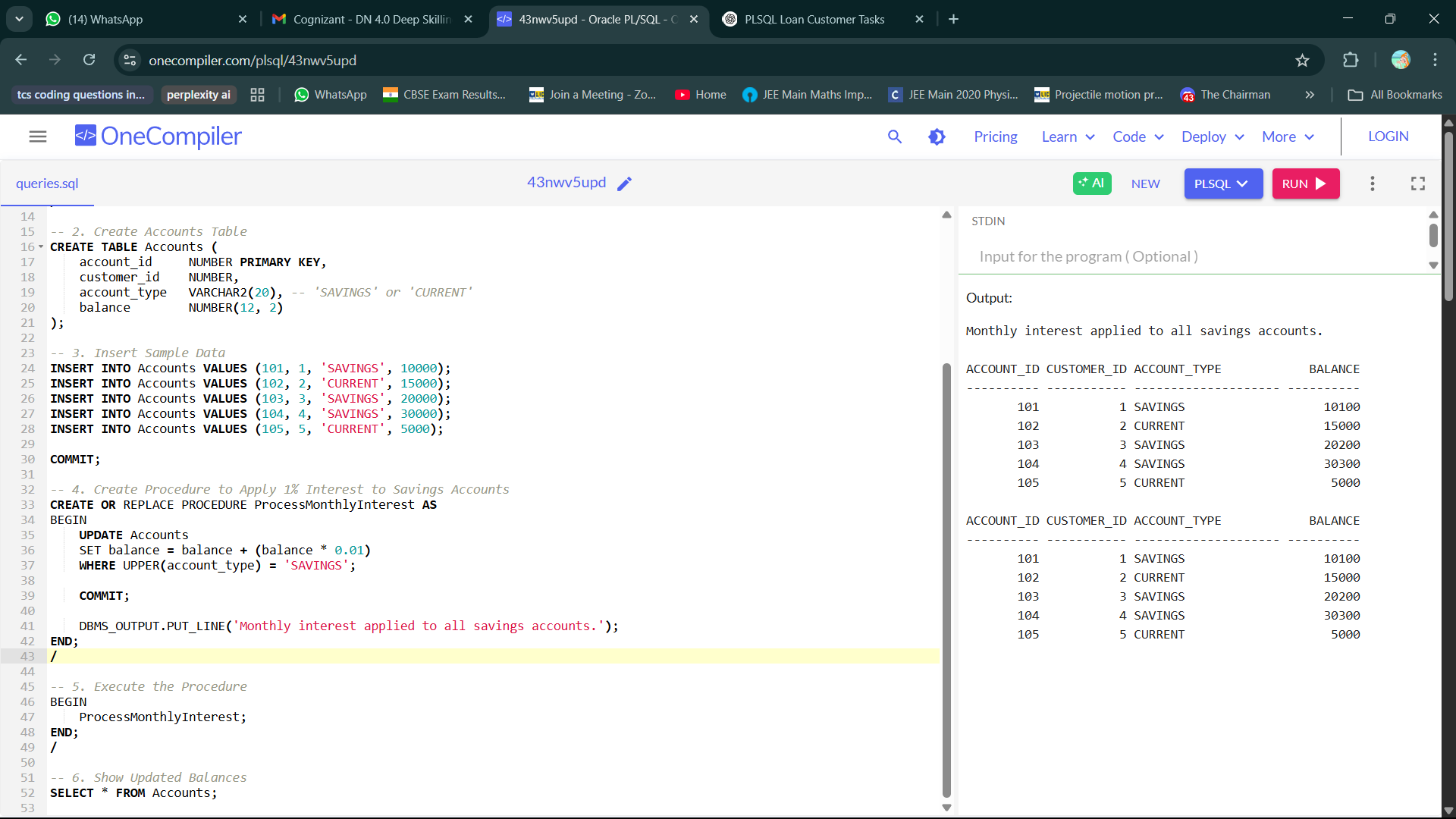
END;

/

SELECT \* FROM Accounts;



* + **Output**



**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.
  + **Code**

CREATE UpdateEmployeeBonus(

deptName IN VARCHAR2,

bonusPercent IN NUMBER

) AS

v\_count NUMBER;

BEGIN

-- Update employee salaries

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE Department = deptName;

-- Get number of affected rows

v\_count := SQL%ROWCOUNT;

-- Print message

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonusPercent || '% applied to ' || v\_count ||

' employee(s) in department "' || deptName || '".');

COMMIT;

END;

/

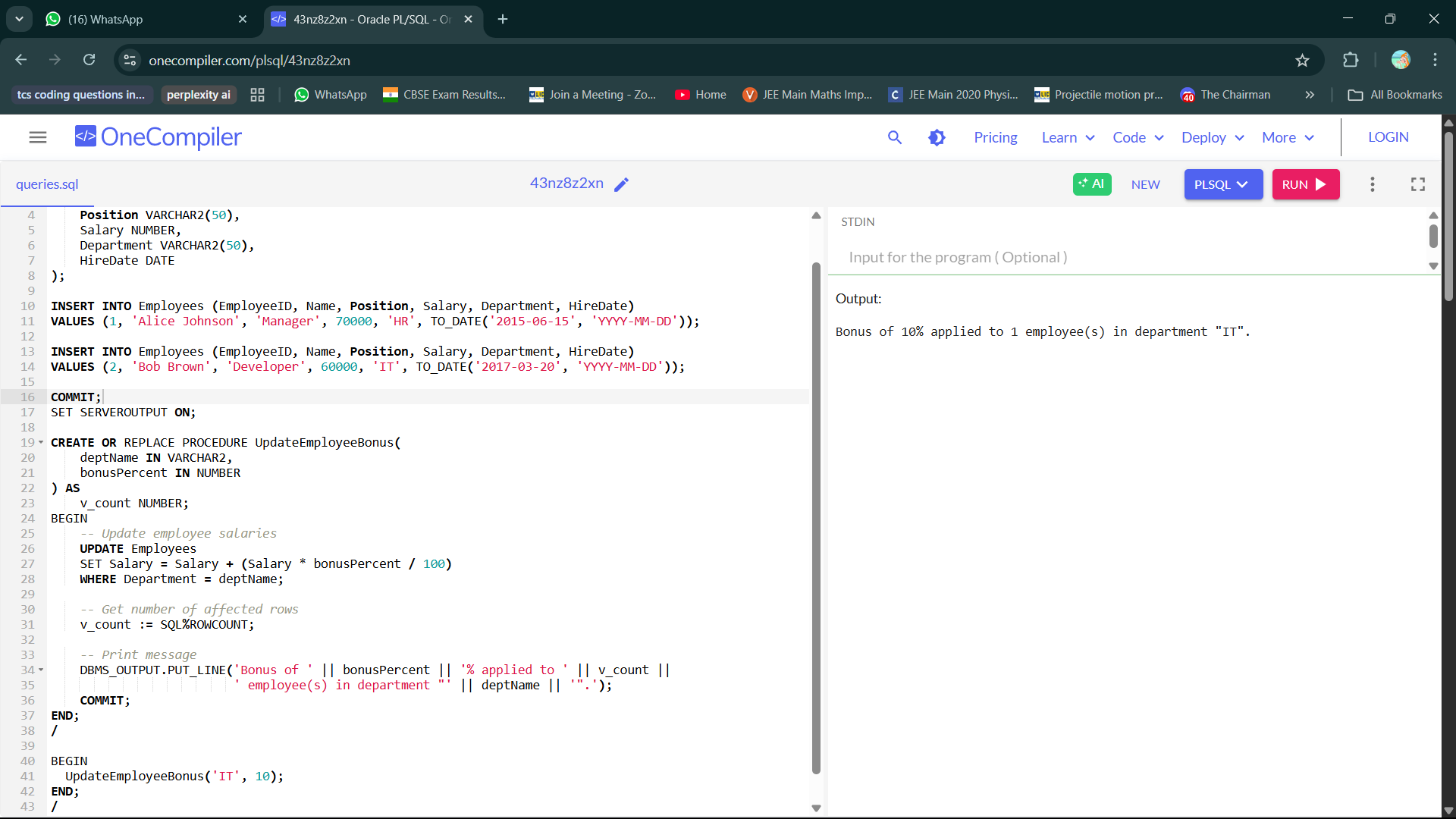
BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

**Output**



**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.
  + **Code**

CREATE TABLE Accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20),

balance NUMBER(12, 2)

);

INSERT INTO Accounts VALUES (201, 101, 'SAVINGS', 10000);

INSERT INTO Accounts VALUES (202, 102, 'CURRENT', 8000);

INSERT INTO Accounts VALUES (203, 103, 'SAVINGS', 12000);

SELECT \* from Accounts;

COMMIT;

CREATE PROCEDURE TransferFunds (

p\_source\_account\_id IN NUMBER,

p\_target\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_source\_balance NUMBER(12,2);

BEGIN

SELECT balance INTO v\_source\_balance

FROM Accounts

WHERE account\_id = p\_source\_account\_id

FOR UPDATE;

IF v\_source\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

UPDATE Accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_source\_account\_id;

UPDATE Accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_target\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(p\_amount || ' transferred from account ' || p\_source\_account\_id ||

' to account ' || p\_target\_account\_id);

END;

/

BEGIN

TransferFunds(201, 202, 3000); --3000 transferred from 201 to 202

END;

/

-- 5. Check updated account balances

SELECT \* FROM Accounts;

* + **Output**

A screenshot of a computer

AI-generated content may be incorrect.