Department of Computer Science and Engineering

Islamic University of Technology (IUT)

A subsidiary organ of OIC

Lab-7 Report

CSE 4508: RDBMS Lab Report

Name: Anm Zahid Hossain Milkan

Student ID: 200041202

Section: 2B

Semester: 5th

Academic Year: 2022-23

Date of Submission: 17/10/23

Task A: A software company has a table of developers with their ID, name, salary, and years of

experience.

Write a PL/SQL block to update developer salaries based on experience:

● Increase salary by 8% for developers with 2-4 years experience

● Increase salary by 12% for developers with 5-7 years experience

● Increase salary by 15% for developers with 8+ years experience

After updating, use an implicit cursor to display the number of salaries changed.

Solution:

```
--Task A--
CREATE TABLE developers (
    ID NUMBER PRIMARY KEY,
    name VARCHAR2(50),
    salary NUMBER,
    years_of_experience NUMBER
);

-- Insert
INSERT INTO developers (ID, name, salary,
years_of_experience)
VALUES (1, 'rafin', 50000, 3);

INSERT INTO developers (ID, name, salary,
years_of_experience)
VALUES (2, 'amit', 60000, 6);
```

```
INSERT INTO developers (ID, name, salary,
years_of_experience)
VALUES (3, 'sahab', 70000, 9);

INSERT INTO developers (ID, name, salary,
years_of_experience)
VALUES (4, 'zahid', 70000, 10);
COMMIT;

INSERT INTO developers (ID, name, salary,
years_of_experience)
VALUES (5, 'zahid', 70000, 1);
COMMIT;
--procedure
DECLARE
    updated_count NUMBER := 0;
    count1 NUMBER :=0;
    count2 NUMBER :=0;
    count3 NUMBER :=0;

BEGIN

    FOR dev_rec IN (SELECT ID, name, salary,
years_of_experience FROM developers) LOOP

        IF dev_rec.years_of_experience BETWEEN 2 AND 4
THEN
            UPDATE developers
            SET salary = dev_rec.salary * 1.08
            WHERE ID = dev_rec.ID;
            IF SQL%FOUND THEN
```

```
            count1 :=SQL%ROWCOUNT;
            END IF;
        ELSIF dev_rec.years_of_experience BETWEEN 5 AND 7
THEN
            UPDATE developers
            SET salary = dev_rec.salary * 1.12
            WHERE ID = dev_rec.ID;
            IF SQL%FOUND THEN
            count2 :=SQL%ROWCOUNT;
             END IF;
        ELSIF dev_rec.years_of_experience >= 8 THEN
            UPDATE developers
            SET salary = dev_rec.salary * 1.15
            WHERE ID = dev_rec.ID;
            IF SQL%FOUND THEN
            count3 :=SQL%ROWCOUNT;
             END IF;

        END IF;
        updated_count :=count1+count2+count3;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Number of salaries changed: '
|| updated_count);

    COMMIT;
END;
/
```

Output:

```
 41  /
Number of salaries changed: 3

PL/SQL procedure successfully completed.
```

Explanation:

In this code, I began by creating a database table called "developers" to store information about software developers. After defining the table structure, I inserted sample data for four developers into the table. The main part of the code consists of a PL/SQL procedure. This procedure iterates through the "developers" table, updating developers' salaries based on their years of experience. If a developer has 2 to 4 years of experience, their salary is increased by 8%. If their experience falls between 5 and 7 years, their salary is raised by 12%. Finally, if a developer has 8 or more years of experience, their salary is boosted by 15%. Throughout the process, I keep track of the number of updated records in each experience category and report the total count of salary changes. This code demonstrates how PL/SQL procedures can efficiently manipulate data within a relational database system, making it a powerful tool for database management.

Task B:

Create a table of transactions (User_ID, Amount, T_Date) that stores all users' bank transactions in a hypothetical bank. Fill up the table with a few transactions of your choice.
Create another table loan_type (Scheme, Installment_Number, Charge, Min_Trans). Loan_type
will have the loan schemes as shown below. For simplicity, you can store the Scheme as a
number, such as 1, 2, or 3 instead of “S-A/S-B/S-C”. Ensure you insert only those 3 specific
rows into the table (Use CHECK constraints). Now, create a function that takes as input a
User_ID, calculates his/her total transactions, and checks against the loan_type table to determine the correct present loan scheme for this person. Determine how an explicit cursor can
be used here and apply it accordingly for this task. The function should return and display the
loan_scheme number.

Scheme	No. of Installment	Service Charge for remaining loan	Eligibility
S-A	30	5%	Total Transaction in the last 12 months ≥ 2000000
S-B	20	10%	Total Transaction in the last 12 months ≥ 1000000
S-C	15	15%	Total Transaction in the last 12 months ≥ 500000

Solution:

```
-- Create the transactions table
CREATE TABLE transactions (
    User_ID NUMBER,
    Amount NUMBER,
    T_Date DATE
);

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (1, 600000, TO_DATE('2023-10-20', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (2, 117500, TO_DATE('2023-10-18', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (3, 2300500, TO_DATE('2023-10-15', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (1, 2145000, TO_DATE('2023-09-25', 'YYYY-MM-DD'));
```

```sql
INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (2, 1600000, TO_DATE('2023-09-23', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (4, 600000, TO_DATE('2023-09-23', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (5, 1100000, TO_DATE('2023-09-23', 'YYYY-MM-DD'));


CREATE TABLE loan_type(
    Scheme NUMBER CHECK (Scheme IN (1, 2, 3)),
    Installment_Number NUMBER,
    Charge NUMBER,
    Min_Trans NUMBER
    );



INSERT INTO loan_type (Scheme, Installment_Number,
Charge, Min_Trans)
VALUES (1, 30, 5, 2000000);

INSERT INTO loan_type (Scheme, Installment_Number,
Charge, Min_Trans)
VALUES (2, 20, 10, 1000000);

INSERT INTO loan_type (Scheme, Installment_Number,
Charge, Min_Trans)
VALUES (3, 15, 15, 500000);
```

```
-- Create a PL/SQL function to determine the loan scheme
CREATE OR REPLACE FUNCTION loan_scheme(p_User_ID IN
NUMBER) RETURN NUMBER IS
    v_total_trans NUMBER := 0;
    v_loan_scheme NUMBER;
    v_loan_min_trans NUMBER;

    CURSOR loan_cursor IS
        SELECT Scheme, Min_Trans
        FROM loan_type;
BEGIN
    SELECT SUM(Amount) INTO v_total_trans
    FROM transactions
    WHERE User_ID = p_User_ID
    AND T_Date >= ADD_MONTHS(SYSDATE, -12);

    v_loan_scheme := 0;

    OPEN loan_cursor;
    FETCH loan_cursor INTO v_loan_scheme,
v_loan_min_trans;

    WHILE loan_cursor%FOUND LOOP
        IF v_total_trans >= v_loan_min_trans THEN
            EXIT;
        END IF;
        FETCH loan_cursor INTO v_loan_scheme,
v_loan_min_trans;
    END LOOP;

    CLOSE loan_cursor;
```

```
    RETURN v_loan_scheme;
END;
/
DECLARE
    scheme_type NUMBER;
BEGIN
    scheme_type := loan_scheme(3);
    IF scheme_type = 1 THEN
        DBMS_OUTPUT.PUT_LINE('Customer is eligable for
Loan service : S-A');
    ELSIF scheme_type = 2 THEN
        DBMS_OUTPUT.PUT_LINE('Customer is eligable for
Loan service : S-B');
    ELSIF scheme_type = 3 THEN
        DBMS_OUTPUT.PUT_LINE('Customer is eligable for
Loan service : S-C');
    END IF;
END;
/
```

Output:

```
SQL> DECLARE
  2      scheme_type NUMBER;
  3  BEGIN
  4      scheme_type := loan_scheme(3);
  5      IF scheme_type = 1 THEN
  6          DBMS_OUTPUT.PUT_LINE('Customer is eligable for Loan service : S-A');
  7      ELSIF scheme_type = 2 THEN
  8          DBMS_OUTPUT.PUT_LINE('Customer is eligable for Loan service : S-B');
  9      ELSIF scheme_type = 3 THEN
 10          DBMS_OUTPUT.PUT_LINE('Customer is eligable for Loan service : S-C');
 11      END IF;
 12  END;
 13  /
Customer is eligable for Loan service : S-A

PL/SQL procedure successfully completed.

SQL>
```

Explanation :

PL/SQL function named "loan_scheme," which takes a User_ID as input and returns the eligible loan scheme. This function performs the following steps:

Calculating Total Transactions: The function calculates the total transaction amount for the given User_ID over the past 12 months. It queries the "transactions" table to obtain this information, ensuring that the transactions occurred within the specified time frame.

Initializing Variables: The function initializes variables to store the selected loan scheme and the minimum transaction required for eligibility. It sets an initial value of 0 for the loan scheme.

Cursor and Loop: A cursor is opened to fetch data from the "loan_type" table, which contains information about different loan schemes and their minimum transaction requirements. The function enters a loop to iterate through these schemes.

Checking Eligibility: Inside the loop, the function checks whether the user's total transactions meet or exceed the minimum transaction requirement for the current loan scheme. If the condition is met, the function sets the loan scheme and exits the loop.

Returning the Loan Scheme: After exiting the loop, the function returns the selected loan scheme based on the user's eligibility.

PL/SQL Block: Following the definition of the "loan_scheme" function, a PL/SQL block is used to demonstrate its functionality. The function is called with a specific User_ID (in this case, User_ID 3), and the result, which represents the eligible loan scheme, is displayed using the DBMS_OUTPUT.PUT_LINE function.