Islamic University of Technology

(CSE 4508: Relational Database Management Systems)

CSE 4508

Lab Report 2

Triggers, Cursors, Recursive Queries and Advanced Aggregation Features

Submitted by:
Abdullah Al Jubaer Gem
Student ID: 210041226
Section: 2B
BSc in CSE, Dept of Computer Science and Engineering

1 Task 1:

In this task, we increase salary of a manager by 10% if his salary is less than 30000. We also decrease salary of an assistant manager by 10% if his salary is more than 20000. We also show how many rows get affected using implicit cursor.

1.1 Affected row count by implicit cursor

```sql
DECLARE
    rows_affected NUMBER := 0;
BEGIN
    UPDATE employees
    SET salary = salary * 1.1
    WHERE designation = 'manager' AND salary < 30000;

    rows_affected := SQL%ROWCOUNT;
    UPDATE employees
    SET salary = salary * 0.9
    WHERE designation = 'assistant manager' AND salary >
   20000;

       rows_affected := rows_affected + SQL%ROWCOUNT;

       DBMS_OUTPUT.PUT_LINE('Total rows affected: ' ||
   rows_affected);
END;
/
```

2 Task 2

In this task, we create a table of transaction and loan type and fill it up with some data. Then we create a function that takes a User ID as input, then calculate his total transactions and check against loan type table to determine the correct loan scheme for that person.

2.1 Get loan scheme by transactions

```sql
CREATE TABLE transactions (
    User_ID INT,
    Amount NUMBER(10, 2),
    T_Date DATE
);
CREATE TABLE loan_type (
    Scheme INT,
    Installment_Number INT,
    Charge NUMBER(5, 2),
    Min_Trans NUMBER(15, 2)
);

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (101, 500000, TO_DATE('2023-01-01', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (102, 1200000, TO_DATE('2023-02-15', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (101, 1800000, TO_DATE('2023-03-10', 'YYYY-MM-DD'));

INSERT INTO transactions (User_ID, Amount, T_Date)
VALUES (103, 700000, TO_DATE('2023-04-20', 'YYYY-MM-DD'));

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 OR REPLACE FUNCTION get_loan_scheme(p_user_id IN
    NUMBER) RETURN NUMBER
IS
    total_transactions NUMBER := 0;
    loan_scheme NUMBER := 0;

    CURSOR c_loan_types IS
        SELECT scheme, min_trans
        FROM loan_type
        ORDER BY min_trans DESC;
BEGIN
    SELECT SUM(amount)
    INTO total_transactions
    FROM transactions
    WHERE user_id = p_user_id;

    FOR loan IN c_loan_types LOOP
        IF total_transactions >= loan.min_trans THEN
            loan_scheme := loan.scheme;
            EXIT;
        END IF;
    END LOOP;

    RETURN loan_scheme;
END;
/
```

3 Task 3:

In this task, we create two triggers first of which automatically initializes a new customer's bill to 0. The other one updates the customer's bill based on Pricing plan after each phone call.

3.1 Initializing new customer bill

```
CREATE OR REPLACE TRIGGER new_customer_bill_initialize
AFTER INSERT ON customer
FOR EACH ROW
BEGIN
    INSERT INTO bill (SSN, Month, Year, amount)
    VALUES (:NEW.SSN, TO_CHAR(SYSDATE, 'MM'), TO_CHAR(SYSDATE
    , 'YYYY'), 0);
END;
/
);
```

3.2 Customer bill update after each call.

We create a trigger `trg_update_bill_after_call` to automatically update bill based on pricing plan after each phone call.

```
CREATE OR REPLACE TRIGGER update_bill
AFTER INSERT ON phonecall
FOR EACH ROW
DECLARE
    pps NUMBER;
    conn_fee NUMBER;
    call_cost NUMBER;
BEGIN
    SELECT p.PricePerSecond, p.ConnectionFee
    INTO pps, conn_fee
    FROM pricingplan p, customer c
    WHERE c.SSN = :NEW.SSN
      AND c.Plan = p.Code;

    call_cost := conn_fee + (pps * :NEW.Seconds);

    UPDATE bill
    SET amount = amount + call_cost
    WHERE SSN = :NEW.SSN
```

```
END;
/
```

4 Task 4:

In this task, we create a function which generates a student ID automatically for a new student given the department , section , date and program . We then create a procedure that updates all account current balances adding interest if it satisfies some specific conditions.

4.1 Generating the student ID

```
CREATE OR REPLACE FUNCTION Gen_ID (
    date_of_admission DATE,
    dept CHAR,
    prog CHAR,
    sec CHAR
) RETURN VARCHAR2 IS
    new_id VARCHAR2(10);
    year VARCHAR2(2);
    seq VARCHAR2(2);
BEGIN
    year := TO_CHAR(date_of_admission, 'YY');
    seq := student_seq.NEXTVAL;
    new_id := year || '00' || dept || prog || sec || seq;

    RETURN new_id;
END;
/
);
```

4.2 Account balance update

```
CREATE OR REPLACE PROCEDURE update_account_balances IS
    CURSOR acc_cur IS
        SELECT a.ID, a.Balance, ap.interestRate, ap.GP
        FROM Accounts a
        JOIN AccountProperties ap ON a.ID = ap.ID;

    bal Accounts.Balance%TYPE;
```

```sql
    int_rate AccountProperties.interestRate%TYPE;
    gp AccountProperties.GP%TYPE;
    int_amt NUMBER;

BEGIN
    FOR acc_rec IN acc_cur LOOP
        bal := acc_rec.Balance;
        int_rate := acc_rec.interestRate;
        gp := acc_rec.GP;

        IF gp = 1 THEN
            int_amt := (bal * int_rate) / 100;
        ELSIF gp = 2 THEN
            int_amt := (bal * int_rate * 30) / 100;
        ELSIF gp = 3 THEN
            int_amt := (bal * int_rate * 365) / 100;
        ELSE
            int_amt := 0;
        END IF;

        UPDATE Accounts
        SET Balance = Balance + int_amt,
            LastDateofInterest = SYSDATE
        WHERE ID = acc_rec.ID;
    END LOOP;

END update_account_balances;
/
```