LAB 5

Friday, July 14, 2023 8:05 PM

Course Code:	BCSE302P	**Course Name:**	Database Systems Lab
Name:	ASHIMA FATIMA SEIK MUGBUR RAGHMAN	**Department:**	SCOPE
Experiment no:	5	**Registration no:**	21BAI1830
Experiment name:	Procedure & Function		
Faculty(s):	Dr Leninisha Shanmugam		

1. Get two integer values, from which find small and large value and store those values in a two different variables namely small and big and display the output.

```
DECLARE
  small INTEGER;
  big INTEGER;
  input1 INTEGER;
  input2 INTEGER;
BEGIN
  input1 := 10;
  input2 := 20;

  IF input1 < input2 THEN
    small := input1;
    big := input2;
  ELSE
    small := input2;
    big := input1;
  END IF;

  DBMS_OUTPUT.PUT_LINE('Small value: ' || small);
  DBMS_OUTPUT.PUT_LINE('Big value: ' || big);
END;
/
```

```
Statement processed.
Small value: 10
Big value: 20
```

2. Create the tables employee(empid,name,salary,designation,deptid) and department (deptid,name, location, mgrid). Write a PL/SQL program to count the number of employees in each department and check whether the departments having any vacancies or not. Assume that maximum of 45 employees can be placed in each department.

```
CREATE TABLE department (
  deptid NUMBER,
  name VARCHAR2(100),
  location VARCHAR2(100),
  mgrid NUMBER
);

CREATE TABLE employee (
  empid NUMBER,
  name VARCHAR2(100),
  salary NUMBER,
  designation VARCHAR2(100),
  deptid NUMBER
);

INSERT INTO department VALUES (1, 'Department 1', 'Location 1', 101);
INSERT INTO department VALUES (2, 'Department 2', 'Location 2', 102);

INSERT INTO employee VALUES (1, 'Employee 1', 5000, 'Designation 1', 1);
INSERT INTO employee VALUES (2, 'Employee 2', 6000, 'Designation 2', 1);
INSERT INTO employee VALUES (3, 'Employee 3', 7000, 'Designation 3', 1);
INSERT INTO employee VALUES (4, 'Employee 4', 8000, 'Designation 4', 2);
```

```sql
SELECT * FROM department;
SELECT * FROM employee;
```

DEPTID	NAME	LOCATION	MGRID
1	Department 1	Location 1	101
2	Department 2	Location 2	102

Download CSV

2 rows selected.

EMPID	NAME	SALARY	DESIGNATION	DEPTID
1	Employee 1	5000	Designation 1	1
2	Employee 2	6000	Designation 2	1
3	Employee 3	7000	Designation 3	1
4	Employee 4	8000	Designation 4	2

```sql
DECLARE
  dept_count NUMBER;
  vacancies BOOLEAN;
BEGIN
  FOR dept_rec IN (SELECT deptid, name FROM department) LOOP
    SELECT COUNT(*) INTO dept_count FROM employee WHERE deptid = dept_rec.deptid;

    IF dept_count >= 45 THEN
      vacancies := FALSE;
    ELSE
      vacancies := TRUE;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Department: ' || dept_rec.name);
    DBMS_OUTPUT.PUT_LINE('Number of Employees: ' || TO_CHAR(dept_count));
    DBMS_OUTPUT.PUT_LINE('Vacancies Available: ' || CASE WHEN vacancies THEN 'Yes' ELSE 'No' END);
  END LOOP;
END;
/
```

```
Statement processed.
Department: Department 1
Number of Employees: 3
Vacancies Available: Yes
Department: Department 2
Number of Employees: 1
Vacancies Available: Yes
```

3. Write a PL/SQL procedure to calculate the incentive amount given for each employee if 10% incentive of salary is provided.

```sql
CREATE OR REPLACE PROCEDURE calculate_incentive AS
  incentive_amt NUMBER;
BEGIN
  FOR emp_rec IN (SELECT empid, salary FROM employee) LOOP
    incentive_amt := emp_rec.salary * 0.1;

    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_rec.empid);
    DBMS_OUTPUT.PUT_LINE('Incentive Amount: ' || incentive_amt);
  END LOOP;
END;
/

EXECUTE calculate_incentive;
```

```
Procedure created.

Statement processed.
Employee ID: 1
Incentive Amount: 500
Employee ID: 2
Incentive Amount: 600
Employee ID: 3
Incentive Amount: 700
Employee ID: 4
Incentive Amount: 800
```

4. Create a table stock to contain the item Code, item name, current stock and date of last purchase. Write a stored procedure to seek for an item using item code & delete if, if the date of last purchased is before one year from the current date. If not update the current stock.

```sql
CREATE TABLE stock_21BAI1830 (
  item_code NUMBER,
  item_name VARCHAR2(100),
  current_stock NUMBER,
  last_purchase_date DATE
);

INSERT INTO stock_21BAI1830 VALUES (1, 'Item A', 50, TO_DATE('2022-07-01', 'YYYY-MM-DD'));
INSERT INTO stock_21BAI1830 VALUES (2, 'Item B', 30, TO_DATE('2023-01-15', 'YYYY-MM-DD'));
INSERT INTO stock_21BAI1830 VALUES (3, 'Item C', 20, TO_DATE('2023-07-10', 'YYYY-MM-DD'));

SELECT * FROM stock_21BAI1830;
```

```
1 row(s) inserted.
```

ITEM_CODE	ITEM_NAME	CURRENT_STOCK	LAST_PURCHASE_DATE
1	Item A	50	01-JUL-22
2	Item B	30	15-JAN-23
3	Item C	20	10-JUL-23

```
CREATE OR REPLACE PROCEDURE delete_or_update_item_21BAI1830(item_code_param IN NUMBER) AS
  last_purchase_date_var DATE;
BEGIN
  SELECT last_purchase_date INTO last_purchase_date_var
  FROM stock_21BAI1830
  WHERE item_code = item_code_param;

  IF last_purchase_date_var < ADD_MONTHS(TRUNC(SYSDATE), -12) THEN
    DELETE FROM stock_21BAI1830 WHERE item_code = item_code_param;
    DBMS_OUTPUT.PUT_LINE('Item deleted successfully.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Item stock updated.');
  END IF;
END;
/

EXECUTE delete_or_update_item_21BAI1830(2);
```

```
Procedure created.

Statement processed.
Item stock updated.
```

5. Create a table contain phone number, user name and address of the phone user. Write the function to search for address using phone number.

```
CREATE TABLE phone_directory (
  phone_number NUMBER,
  user_name VARCHAR2(100),
  address VARCHAR2(200)
);

INSERT INTO phone_directory VALUES (1234567890, 'User 1', 'Address 1');
INSERT INTO phone_directory VALUES (9876543210, 'User 2', 'Address 2');

SELECT * FROM phone_directory;
```

```
Table created.

1 row(s) inserted.

1 row(s) inserted.
```

PHONE_NUMBER	USER_NAME	ADDRESS
1234567890	User 1	Address 1
9876543210	User 2	Address 2

```sql
CREATE OR REPLACE FUNCTION search_address(phone_number_param IN NUMBER) RETURN VARCHAR2 AS
  address_var VARCHAR2(200);
BEGIN
  SELECT address INTO address_var
  FROM phone_directory
  WHERE phone_number = phone_number_param;

  RETURN address_var;
END;
/

DECLARE
  address_result VARCHAR2(200);
BEGIN
  address_result := search_address(1234567890);
  DBMS_OUTPUT.PUT_LINE('Address: ' || address_result);
END;
/
```

```
Function created.


Statement processed.
Address: Address 1
```

LAB 6

Friday, July 14, 2023 8:06 PM

Course Code:	BCSE302P	**Course Name:**	Database Systems Lab
Name:	ASHIMA FATIMA SEIK MUGBUR RAGHMAN	**Department:**	SCOPE
Experiment no:	6	**Registration no:**	21BAI1830
Experiment name:	Cursors and Triggers		
Faculty(s):	Dr Leninisha Shanmugam		

```
CREATE TABLE EMP_21BAI1830 (
  Empno NUMBER(4),
  Ename VARCHAR2(30),
  Job VARCHAR2(15),
  Sal NUMBER(8,2),
  Dept NUMBER(2),
  Commission NUMBER(7,2)
);

INSERT INTO EMP_21BAI1830 (Empno, Ename, Job, Sal, Dept, Commission) VALUES (1, 'John', 'Manager', 5000, 10, NULL);
INSERT INTO EMP_21BAI1830 (Empno, Ename, Job, Sal, Dept, Commission) VALUES (2, 'Jane', 'Analyst', 4000, 10, NULL);
INSERT INTO EMP_21BAI1830 (Empno, Ename, Job, Sal, Dept, Commission) VALUES (3, 'Mark', 'Clerk', 3000, 20, NULL);
INSERT INTO EMP_21BAI1830 (Empno, Ename, Job, Sal, Dept, Commission) VALUES (4, 'Mary', 'Clerk', 2000, 20, NULL);
INSERT INTO EMP_21BAI1830 (Empno, Ename, Job, Sal, Dept, Commission) VALUES (5, 'Adam', 'Manager', 6000, 30, NULL);
```

EMPNO	ENAME	JOB	SAL	DEPT	COMMISSION
1	John	Manager	5000	10	-
2	Jane	Analyst	4000	10	-
3	Mark	Clerk	3000	20	-
4	Mary	Clerk	2000	20	-
5	Adam	Manager	6000	30	-

I. (a) Write a PL/SQL code to display the Empno, Ename and Job of employees of DeptNo 10 with CURSOR FOR LOOP Statement.

```
DECLARE
  CURSOR c_employees IS
    SELECT Empno, Ename, Job
    FROM EMP_21BAI1830
    WHERE Dept = 10;
BEGIN
  FOR emp_rec IN c_employees LOOP
    DBMS_OUTPUT.PUT_LINE('Empno: ' || emp_rec.Empno || ', Ename: ' || emp_rec.Ename || ', Job: ' || emp_rec.Job);
  END LOOP;
END;
/
```

```
Statement processed.
Empno: 1, Ename: John, Job: Manager
Empno: 2, Ename: Jane, Job: Analyst
```

(b) Create a Cursor to increase the salary of employees according to the following conditions:

- Salary of DeptNo 10 employees increased by 1000.
- Salary of DeptNo 20 employees increased by 500.
- Salary of DeptNo 30 employees increased by 800.

Also store the EmpNo, old salary and new salary in a Table TEMP having three columns Empid, Old and New.

```sql
CREATE TABLE TEMP (
  Empid NUMBER,
  Old NUMBER,
  New NUMBER
);

DECLARE
  CURSOR c_employees IS
    SELECT Empno, Sal, Dept
    FROM EMP_21BAI1830;
BEGIN
  FOR emp_rec IN c_employees LOOP
    IF emp_rec.Dept = 10 THEN
      INSERT INTO TEMP VALUES (emp_rec.Empno, emp_rec.Sal, emp_rec.Sal + 1000);
    ELSIF emp_rec.Dept = 20 THEN
      INSERT INTO TEMP VALUES (emp_rec.Empno, emp_rec.Sal, emp_rec.Sal + 500);
    ELSIF emp_rec.Dept = 30 THEN
      INSERT INTO TEMP VALUES (emp_rec.Empno, emp_rec.Sal, emp_rec.Sal + 800);
    END IF;
  END LOOP;

  COMMIT;
END;
```

```
Table created.
```

```sql
SELECT * FROM TEMP;
```

EMPID	OLD	NEW
1	5000	6000
2	4000	5000
3	3000	3500
4	2000	2500
5	6000	6800

c) Write a program in PL/SQL to create a cursor displays the name and salary of each employee in the EMPLOYEES table whose salary is less than average salary of all employees.

```
DECLARE
  avg_salary NUMBER;

  CURSOR c_employees IS
    SELECT Ename, Sal
    FROM EMP_21BAI1830
    WHERE Sal < avg_salary;
BEGIN
  SELECT AVG(Sal) INTO avg_salary FROM EMP_21BAI1830;

  FOR emp_rec IN c_employees LOOP
    DBMS_OUTPUT.PUT_LINE('Name: ' || emp_rec.Ename || ', Salary: ' || emp_rec.Sal);
  END LOOP;
END;
/
```

```
Statement processed.
Name: Mark, Salary: 3000
Name: Mary, Salary: 2000
```

II. (a) Create a trigger, which verify that updated salary of employee must be greater than his/her previous salary

```
CREATE OR REPLACE TRIGGER check_salary_increase
BEFORE UPDATE ON EMP_21BAI1830
FOR EACH ROW
BEGIN
  IF :NEW.Sal <= :OLD.Sal THEN
    raise_application_error(-20001, 'Salary must be greater than the previous salary.');
  END IF;
END;
/
```

```
Trigger created.
```

```sql
UPDATE EMP_21BAI1830 SET Sal = 5500 WHERE Empno = 1;
UPDATE EMP_21BAI1830 SET Sal = 3500 WHERE Empno = 2;
```

```
1 row(s) updated.

ORA-20001: Salary must be greater than the previous salary. ORA-06512: at
"SQL_MTOLRSVMBNFKJGTUYUNWLVKCX.CHECK_SALARY_INCREASE", line 3
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

(b) Create a trigger, which verify that for the department 2, the number of employees should not exceed than 5.

```sql
CREATE OR REPLACE TRIGGER check_employee_count
BEFORE INSERT OR UPDATE ON EMP_21BAI1830
FOR EACH ROW
DECLARE
  employee_count NUMBER;
BEGIN
  IF :NEW.Dept = 2 THEN
    SELECT COUNT(*) INTO employee_count
    FROM EMP_21BAI1830
    WHERE Dept = 2;

    IF employee_count >= 5 THEN
      raise_application_error(-20002, 'Maximum employee count reached for department 2.');
    END IF;
  END IF;
END;
/
```

```
Trigger created.
```

EMPNO	ENAME	JOB	SAL	DEPT	COMMISSION
7	Robert	Clerk	2500	2	-
8	Emma	Analyst	3500	2	-
9	Michael	Manager	5500	2	-
10	Sophia	Clerk	2800	2	-
1	John	Manager	5500	10	-
2	Jane	Analyst	4000	10	-
3	Mark	Clerk	3000	20	-
4	Mary	Clerk	2000	20	-
5	Adam	Manager	6000	30	-
6	Alex	Analyst	4500	2	-

```
INSERT INTO EMP_21BAI1830 (Empno, Ename, Job, Sal, Dept, Commission) VALUES (11, 'Oliver', 'Analyst', 4200, 2, NULL);
-- This insert will fail with an error because the department 2 already has 5 employees
```

```
ORA-20002: Maximum employee count reached for department 2. ORA-06512: at "SQL_MTOLRSVMBNFKJGTUYUNWLVKCX.CHECK_EMPLOYEE_COUNT", line 10
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

(c) Utilize Account table (referred as t) and loan table(referred as s) and Create the trigger for negative balance as mentioned as below.

i) Insert a new tuple s in the loan relation with
s[loan_no] = t[acctno]
s[br_name] = t[br_name]
s[amount] = -t[balance]

ii) Insert a new tuple u in the borrower relation with
u[custname] = ”Jones”
u[loan_no] = t[acctno]

iii) Set t[balance] to 0.

```
CREATE TABLE Loan (
  loan_no NUMBER,
  br_name VARCHAR2(100),
  amount NUMBER
);

CREATE TABLE Borrower (
  custname VARCHAR2(100),
  loan_no NUMBER
);
```

```
CREATE TABLE Account (
  acctno NUMBER,
  br_name VARCHAR2(100),
  balance NUMBER
);
```

```sql
CREATE TABLE Loan (
  loan_no NUMBER,
  br_name VARCHAR2(100),
  amount NUMBER
);

CREATE TABLE Borrower (
  custname VARCHAR2(100),
  loan_no NUMBER
);
```

```sql
CREATE TABLE Account (
  acctno NUMBER,
  br_name VARCHAR2(100),
  balance NUMBER
);
```

Table created.

Table created.

```sql
CREATE OR REPLACE TRIGGER negative_balance_trigger
BEFORE UPDATE OF balance ON Account
FOR EACH ROW
DECLARE
  loan_no_temp NUMBER;
BEGIN
  IF :NEW.balance < 0 THEN
    SELECT loan_no INTO loan_no_temp FROM Loan WHERE loan_no = :OLD.acctno;

    IF loan_no_temp IS NULL THEN
      INSERT INTO Loan (loan_no, br_name, amount) VALUES (:OLD.acctno, :OLD.br_name, -1 * :NEW.balance);
    END IF;

    INSERT INTO Borrower (custname, loan_no) VALUES ('Jones', :OLD.acctno);

    :NEW.balance := 0;
  END IF;
END;
/
```

Trigger created.

```sql
INSERT INTO Account (acctno, br_name, balance) VALUES (1, 'Branch A', 1000);
INSERT INTO Account (acctno, br_name, balance) VALUES (2, 'Branch B', -500);
INSERT INTO Account (acctno, br_name, balance) VALUES (3, 'Branch C', -200);
INSERT INTO Account (acctno, br_name, balance) VALUES (4, 'Branch D', 300);

INSERT INTO Loan (loan_no, br_name, amount) VALUES (1, 'Branch A', 500);
INSERT INTO Loan (loan_no, br_name, amount) VALUES (2, 'Branch B', 200);


UPDATE Account SET balance = -200 WHERE acctno = 1;
-- This update will trigger the trigger because the new balance is negative (-200)
-- The trigger will insert a new tuple into the Loan table and set the balance to 0
-- Also, a new tuple will be inserted into the Borrower table

SELECT * FROM Loan;
SELECT * FROM Borrower;
```

1 row(s) updated.

LOAN_NO	BR_NAME	AMOUNT

CUSTNAME	LOAN_NO

1 row(s) updated.

LOAN_NO	BR_NAME	AMOUNT
1	Branch A	500
2	Branch B	200

CUSTNAME	LOAN_NO
Jones	1