LAB SESSION 2
(TRIGGERS)

EXERCISE 1

1- Create the following table and trigger;

```
CREATE TABLE EMPLOYEES (
    EMP_ID NUMBER PRIMARY KEY,
    EMP_NAME VARCHAR2(50),
    HIRE_DATE DATE,
    SALARY NUMBER
);
/
CREATE OR REPLACE TRIGGER TRG_PREVENT_LOW_SALARY
BEFORE INSERT OR UPDATE ON EMPLOYEES
FOR EACH ROW
BEGIN
    IF :NEW.SALARY < 3000 THEN
        DBMS_OUTPUT.PUT_LINE('Salary must be at least 3000.');
    END IF;
END;
```

2- Run and analyze the output of the execution of the inserting statement.

```
INSERT INTO EMPLOYEES (EMP_ID, EMP_NAME, HIRE_DATE, SALARY) VALUES (1,
'Jane Doe', SYSDATE, 2500);
```

3- Replace the line **`DBMS_OUTPUT.PUT_LINE …;`** by the following line and test again the trigger (by inserting new record)

```
RAISE_APPLICATION_ERROR(-20001, 'Salary must be at least 3000.');
```

4- Explain the keys: *NEW, RAISE_APPLICATION_ERROR, -20001*

5- Add new trigger that print in (DBMS OUTPLUT) the old salary of an employee after modifying it.

EXERCISE 2

1- Design a table named EMPLOYEE_LOG with the following columns:

- LOG_ID: A unique identifier for each log entry.
- EMP_ID: The ID of the employee whose record was changed.
- OPERATION: The type of operation performed (UPDATE or DELETE).
- OLD_SALARY: The salary of the employee before the change (if applicable).
- CHANGE_DATE: The timestamp of the change.

2- Write the SQL statement to create this table.

3- Write a trigger named **`TRG_EMPLOYEE_LOG`** that:

4- Activates after **`UPDATE`** or **`DELETE`** operations on the **`EMPLOYEES`** table.

5- Inserts a new row into the **`EMPLOYEE_LOG`** table capturing:

   - The **`EMP_ID`**.
   - The **`OPERATION`** type (**`UPDATE`** or **`DELETE`**).
   - The **`OLD_SALARY`** before the change.
   - The current timestamp (**`SYSDATE`**).