Islamic University of Technology (IUT)

CSE 4308: Database Management Systems Lab

Lab Report # 10

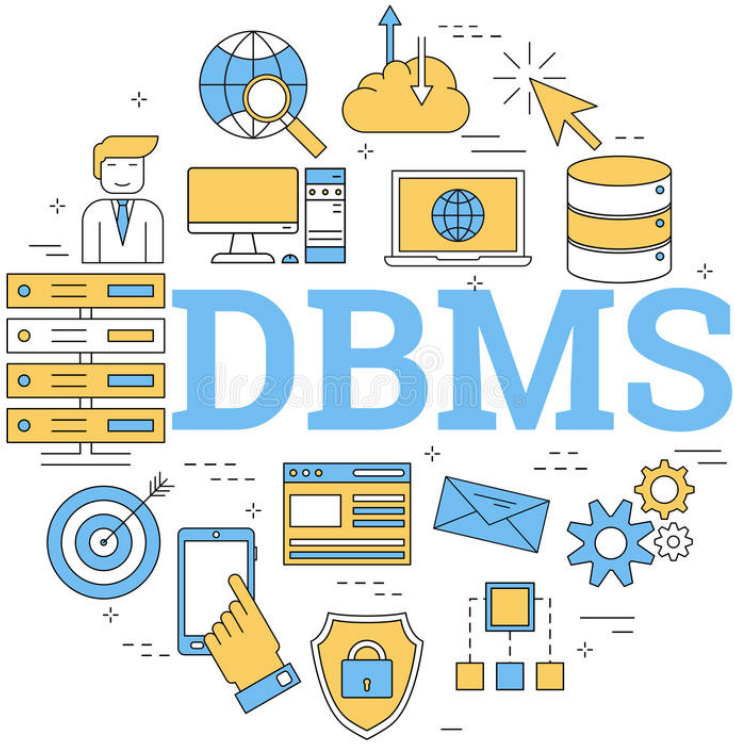
Submitted to:

Md. Bakhtiar Hasan,

Asst. Professor, CSE.

Zannatun Naim Srsity,

Lecturer, CSE.



Submitted by:

M M Nazmul Hossain
ID 200042118
CSE (SWE)

Submission Date:

16.11.2022

Introduction

The tenth Database Management Systems Lab was about Advanced SQL. PL/SQL or Procedural Language extension to Structured Query Language, can simplify application development and optimize its execution. It removes the dependency on OJDBC, rather is tightly integrated with SQL itself. In this lab, more features of PL/SQL were explored like Cursors and Triggers.

Method

At first, the provided DDL+drop.sql file is imported into the Database to create the tables and the smallRelationsInsertFile.sql is imported to insert the values in the tables, which will be required for this lab. Before completing any tasks, the server output size has to be set which can be done by writing:

```
SET SERVEROUTPUT ON SIZE 1000000
```

Analysis of the problem of Task 1

The first task was to implement an update statement and determine the number of rows which weren't affected by the update statement.

The Code

```
-- TASK 1
DECLARE
    TOTAL_ROWS    NUMBER(10);
    AFFECTED_ROWS NUMBER(10);
BEGIN
    UPDATE DEPARTMENT
    SET
        BUDGET = BUDGET - (
            BUDGET*0.1
        )
    WHERE
        BUDGET > 99999;
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO BUDGET WAS MORE THAN THE GIVEN LIMIT');
    ELSIF SQL%FOUND THEN
        AFFECTED_ROWS:= SQL%ROWCOUNT;
        SELECT
```

```
            COUNT(DEPT_NAME) INTO TOTAL_ROWS
        FROM
            DEPARTMENT;
        TOTAL_ROWS := TOTAL_ROWS - AFFECTED_ROWS;
        DBMS_OUTPUT.PUT_LINE(TOTAL_ROWS
            || ' ROWS WERE NOT UPDATED');
    END IF;
END;
/
```

Explanation of Solution

- First, two variables were declared, Total_Rows and Affected_Rows.
- The SQL query for the necessary update function is executed.
- Then the number of rows affected by the update statement is stored in the Affected_Rows variable using SQL%ROWCOUNT.
- In case none of the rows were affected, SQL%NOUTFOUND and SQL%FOUND are used in if conditions to avoid errors.
- An SQL query is run that stores the total number of rows in the budget attribute into the variable Total_Rows.
- Finally, to determine the number of unaffected rows, Total_Rows is subtracted by Affected_Rows, and then that value is displayed as the final result.

Analysis of the problem of Task 2

Task 2 was to create a procedure that will take the day of the week, start time, and end time as parameters and return the name of the instructors who would be taking a class during that time period.

The Code

```
-- TASK 2
CREATE OR REPLACE PROCEDURE CURR_INSTRUCTORS(
    DAY_FIRST_LETTER IN TIME_SLOT.DAY%TYPE,
    START_TIME IN TIME_SLOT.START_HR%TYPE,
    END_TIME IN TIME_SLOT.END_HR%TYPE
) AS
    CURSOR INS_INFO IS
        SELECT
```

```
            INSTRUCTOR.NAME AS NAME,
            DAY,
            START_HR,
            START_MIN,
            END_HR,
            END_MIN
        FROM
            INSTRUCTOR
            NATURAL JOIN TEACHES
            NATURAL JOIN SECTION
            NATURAL JOIN TIME_SLOT
        WHERE
            DAY = DAY_FIRST_LETTER
            AND START_HR >= START_TIME
            AND END_HR <= END_TIME;

BEGIN
    FOR I IN INS_INFO LOOP
        DBMS_OUTPUT.PUT_LINE(I.NAME);
    END LOOP;
END;
/

DECLARE
    DAY_FIRST_LETTER TIME_SLOT.DAY%TYPE;
    START_TIME       TIME_SLOT.START_HR%TYPE;
    END_TIME         TIME_SLOT.END_HR%TYPE;
BEGIN
    DAY_FIRST_LETTER := '&DAY_FIRST_LETTER';
    START_TIME :='&START_TIME';
    END_TIME:='&END_TIME';
    CURR_INSTRUCTORS(DAY_FIRST_LETTER, START_TIME, END_TIME);
END;
/
```

Explanation of Solution

- A Procedure is created based on the given requirements. The %TYPE instruction is used to more accurately assign the domain of the parameters.

- A cursor is declared which completes all the necessary queries. If any teacher was taking any class in between the mentioned time period, their name would be displayed.
- The cursor is traversed and its value is displayed.
- In the anonymous block, the procedure is called providing the required variables, after taking input from the user.

Analysis of the problem of Task 3

Task 3 was to write a procedure that will take a number N in as a parameter. It will return the top N students based on the number of courses they are enrolled in.

The Code

```
-- TASK 3
CREATE OR REPLACE PROCEDURE STUDENT_INFORMATION(
    NUM IN NUMBER
) AS
    CURSOR STD_INFO IS
        SELECT
            *
        FROM
            (
                SELECT
                    ID,
                    MAX(NAME)        AS NAME,
                    MAX(DEPT_NAME)   AS DEPT_NAME,
                    COUNT(COURSE_ID) AS NO_OF_COURSES
                FROM
                    STUDENT
                    NATURAL JOIN TAKES
                    NATURAL JOIN SECTION
                    NATURAL JOIN COURSE
                GROUP BY
                    ID
                ORDER BY
                    NO_OF_COURSES DESC
            )
        WHERE
            ROWNUM <=NUM;
BEGIN
```

```
    FOR I IN STD_INFO LOOP
        DBMS_OUTPUT.PUT_LINE(I.ID
            ||CHR(9)
            ||I.NAME
            ||CHR(9)
            ||I.DEPT_NAME
            ||CHR(9)
            ||I.NO_OF_COURSES);
    END LOOP;
END;
/

DECLARE
    NUM NUMBER;
BEGIN
    NUM :='&NUM';
    STUDENT_INFORMATION(NUM);
END;
/
```

Explanation of Solution

- The procedure is created according to the given requirements with parament NUM.
- A cursor is declared which counts course id grouped by ID to determine the no of courses taken by each student.
- It is ordered by descending keeping students with more number of courses on top.
- The entire query is put in a nested query, and all value is selected where the rownum is less than or equal to the input parameter NUM.
- The values in the cursor are traversed using the for loop and displayed.
- In an anonymous block, take input NUM from the user and call the Student_Information function.

Analysis of the problem of Task 4

Task 4 was to generate an Auto incrementing Id attribute which is automatically assigned based on the insert.

The Code

```
-- TASK 4
CREATE SEQUENCE INSTRUCTOR_ID
MINVALUE 10001
MAXVALUE 99999
START WITH 10001
INCREMENT BY 1
CACHE 20;

CREATE OR REPLACE TRIGGER AUTO_INCREMENT BEFORE
    INSERT ON INSTRUCTOR FOR EACH ROW
BEGIN
    :NEW.ID := INSTRUCTOR_ID.NEXTVAL;
END;
/
INSERT INTO INSTRUCTOR VALUES (
    '0',
    'Nazmul',
    'Finance',
    '69420'
);
INSERT INTO INSTRUCTOR VALUES (
    '0',
    'Sian',
    'Comp. Sci.',
    '42069'
);
SELECT * FROM INSTRUCTOR;
```

Explanation of Solution

- A Sequence for ID is at first generated. The max value, min value, increment value, and cache are set.
- A trigger is created that will activate before inserting it on the instructor table for each row. It will assign the automatically generated id to that record.
- During Insert, a dummy value is input in the Instructor ID like 0.
- The Select statement is used to verify whether the implemented trigger worked properly.

Analysis of the problem of Task 5

Task 5 was to assign advisors to students based on student insert, automatically, depending on the number of students assigned to each advisor.

The Code

```
-- TASK 5
CREATE OR REPLACE TRIGGER SET_ADVISOR AFTER
    INSERT ON STUDENT FOR EACH ROW
DECLARE
    I_ID INSTRUCTOR.ID%TYPE;
BEGIN
    SELECT
        ID INTO I_ID
    FROM
        (
            SELECT
                INSTRUCTOR.ID,
                INSTRUCTOR.NAME,
                COUNT(ADVISOR.S_ID) AS NUM_OF_STUDENTS
            FROM
                ADVISOR,
                INSTRUCTOR
            WHERE
                ADVISOR.I_ID = INSTRUCTOR.ID
                AND INSTRUCTOR.DEPT_NAME = :NEW.DEPT_NAME
            GROUP BY
                INSTRUCTOR.ID,
                INSTRUCTOR.NAME
            ORDER BY
                NUM_OF_STUDENTS ASC
        )
    WHERE
        ROWNUM = 1;
    INSERT INTO ADVISOR VALUES(
        :NEW.ID,
        I_ID
    );
END;
/
```

```
INSERT INTO STUDENT VALUES (
    '45320',
    'Naz',
    'Comp. Sci.',
    '108'
);
SELECT * FROM STUDENT;
SELECT* FROM ADVISOR;
```

Explanation of Solution

- A new trigger is created which will activate after inserting in students for each inserted row. A query is run which determines the number of students each advisor has in the department of the newly inserted student. The :NEW. tag is used to differentiate between the dept_names.
- The query is ordered in ascending order so that the advisor with the fewest students is at the top. Then the top row is selected into a local variable. The %TYPE instruction is used to accurately assign the domain of the local variable.
- Finally, new values are inserted into the advisor table, the ID of the newly inserted student, and the ID of the advising instructor.
- A new student is inserted into the student table and both the student and advisor tables are displayed to see whether the trigger is working properly.

Problems

The problems faced during this lab task were:

- Implementing Triggers and keeping every variable in mind is quite taxing and complex. A lot of errors and hurdles had to be crossed to implement them.

- Since most of the topics in this lab was previously discussed or worked on, the lab was quite simple, given enough time.

Findings

PL/SQL allows a lot of things possible in JDBC without having to use Java. This is really convenient as now, the errors can be seen during the compilation, rather than the runtime, more often. PL/SQL Syntax, though quite rigid, is simple once one can hold a grasp on it. Functions and Procedures allow for a much more organized and reliable way of thinking. It makes it easier to manage any database modification. Triggers are an intriguing feature, which allows for a lot of automation of tasks.

Conclusion

Overall, the lab was an invaluable learning experience. It really helped out to understand the core concepts of Advanced SQL. Hopefully, this will allow for a better understanding of Database Management.