Student Grade Report Project Documentation

Student Name: IGIRANEZA Irene

Id: 27328

Date: 25/11/2025

Project: Student Grade Manager using PL/SQL Collections, Records, and GOTO Statements

1. Understanding the Problem

What I Need to Do:

Create a program that takes student test scores and converts them to letter grades, with special recognition for perfect scores.

The Rules:

- **A:** 90-100 points
- **B:** 80-89 points
- **C:** 70-79 points
- **D:** 60-69 points
- **F:** Below 60 points
- **Special:** Students with 100% get a "PERFECT SCORE!" message

2. How I Solved It

Step 1: Setting Up the Data

First, I created a simple table to store student information:

```
sql
CREATE TABLE students (
   student_id NUMBER,
   student_name VARCHAR2(50),
   test_score NUMBER
);
INSERT INTO students VALUES (1, 'Alice', 92);
INSERT INTO students VALUES (2, 'Bob', 78);
INSERT INTO students VALUES (3, 'Charlie', 100);
INSERT INTO students VALUES (4, 'Diana', 85);
INSERT INTO students VALUES (5, 'Evan', 100);
```

Step 2: Understanding the PL/SQL Concepts

What is a RECORD?

I think of a **Record** like a student profile card. Each card holds all information about one student:

text
[Student Card]
Name: Alice
ID: 1
Score: 92
Grade: A

In code:

sql
```
TYPE student_record IS RECORD (
   student_id students.student_id%TYPE,
   student_name students.student_name%TYPE,
   test_score students.test_score%TYPE,
   letter_grade VARCHAR2(2)
);
```

What is a COLLECTION?

A **Collection** is like a class roster that holds all the student cards together:

text
[Class Roster]
1. Alice's Card
2. Bob's Card
3. Charlie's Card
4. Diana's Card
5. Evan's Card

In code:

sql
TYPE student_collection IS TABLE OF student_record;

What is GOTO?

GOTO is like saying "skip to this part" in the instructions. When a student gets 100%, we jump to show a special message.

3. My Complete Solution

sql

```
SET SERVEROUTPUT ON;

DECLARE
   TYPE student_record IS RECORD (
      student_id students.student_id%TYPE,
      student_name students.student_name%TYPE,
      test_score students.test_score%TYPE,
      letter_grade VARCHAR2(2)
   );

   TYPE student_collection IS TABLE OF student_record;

   student_list student_collection;
   v_grade VARCHAR2(2);

BEGIN
   DBMS_OUTPUT.PUT_LINE ('=== STUDENT GRADE REPORT ===');
   DBMS_OUTPUT.PUT_LINE ('');

   SELECT student_id, student_name, test_score, NULL
   BULK COLLECT INTO student_list
   FROM students
   ORDER BY student_id;

   FOR i IN 1..student_list.COUNT LOOP

      DBMS_OUTPUT.PUT ('Student: ' || student_list(i).student_name);
      DBMS_OUTPUT.PUT (' | Score: ' || student_list(i).test_score);
```

```
        CASE
            WHEN student_list(i).test_score >= 90 THEN v_grade := 'A';
            WHEN student_list(i).test_score >= 80 THEN v_grade := 'B';
            WHEN student_list(i).test_score >= 70 THEN v_grade := 'C';
            WHEN student_list(i).test_score >= 60 THEN v_grade := 'D';
            ELSE v_grade := 'F';
        END CASE;

        student_list(i).letter_grade := v_grade;

        DBMS_OUTPUT.PUT (' | Grade: ' || v_grade);

        IF student_list(i).test_score = 100 THEN
            DBMS_OUTPUT.NEW_LINE;
            DBMS_OUTPUT.PUT ('  >>> ');
            GOTO perfect_score_message;
        END IF;

        GOTO next_student;

        <<perfect_score_message>>
        DBMS_OUTPUT.PUT ('PERFECT SCORE! Excellent work!');

        <<next_student>>
        DBMS_OUTPUT.PUT_LINE ('');
        DBMS_OUTPUT.PUT_LINE ('---');

    END LOOP;

    DBMS_OUTPUT.PUT_LINE ('=== GRADE REPORT COMPLETE ===');

END;
/
```

4. What Happens When It Runs

Output:

=== STUDENT GRADE REPORT ===

Student: Alice | Score: 92 | Grade: A

Student: Bob | Score: 78 | Grade: C

Student: Charlie | Score: 100 | Grade: A

>>> PERFECT SCORE! Excellent work!

Student: Diana | Score: 85 | Grade: B

Student: Evan | Score: 100 | Grade: A

>>> PERFECT SCORE! Excellent work!

=== GRADE REPORT COMPLETE ===

PL/SQL procedure successfully completed.

What the Program Does:

1. **Starts** by loading all student data into memory
2. **Loops** through each student one by one
3. **Calculates** the letter grade based on their score
4. **Checks** if the score is 100%
5. **If 100%:** Shows special message using GOTO
6. **If not 100%:** Skips the special message using GOTO
7. **Moves** to the next student

5. What I Learned

Records:

- Help keep related data organized (like a student's information)
- Make code cleaner and easier to understand
- Act like containers for multiple pieces of information about one thing

Collections:

- Let me work with multiple records at once
- Are like arrays or lists in other programming languages
- Make it easy to process all students without writing separate code for each one

GOTO Statements:

- Let me jump to different parts of the code
- Can be useful for simple conditional jumps
- Should be used carefully because they can make code confusing if overused

Alternative to GOTO (Better Practice):

Instead of using GOTO, I could use a simple IF statement:

```
sql
IF student_list(i).test_score = 100 THEN
   DBMS_OUTPUT.PUT_LINE ('');
   DBMS_OUTPUT.PUT_LINE ('  >>> PERFECT SCORE! Excellent work!');
END IF;
```

6. Challenges I Faced

1. **Understanding GOTO:** At first, I was confused about when to use GOTO and how labels work. I practiced with simple examples first.
2. **Collection Syntax:** Remembering to use `student_list.COUNT` instead of a fixed number was tricky at first.
3. **Debugging:** When my output wasn't showing correctly, I learned to add more `DBMS_OUTPUT` statements to see what was happening.

7. How to Test This Project

1. **Run the table creation script** to set up the database
2. **Execute the PL/SQL program** to see the grade report
3. **Try modifying** the test scores in the table and run the program again
4. **Add new students** with different scores to see how it handles them

8. Conclusion

This project helped me understand how to use PL/SQL Records to organize data, Collections to work with multiple items, and GOTO statements for controlling program flow. While GOTO can be useful for simple jumps, I learned that for more complex programs, using IF-THEN-ELSE statements is usually better practice.

The program successfully demonstrates all the required concepts while solving a real-world problem of calculating student grades.