

## Triggers

The following relations are given (primary keys are underlined, optional attributes are denoted with \*):

COURSE(CourseCode, CourseName, Credits)  
STUDENT(RegNum, StudentName, YearFirstEnrollment)  
EXAM\_REGISTRATION(CourseCode, RegNum, Date, Score)  
GRANT\_APPLICATION(RegNum, RequestDate)  
STUDENT\_RANKING(RegNum, TotalPoints)  
GRANT\_AVAILABILITY(Grant#, CourseCode, TeachingHours)  
GRANT\_ASSIGNMENT(Grant#, RegNum, TeachingHours)  
NOTIFICATION(Not#, Grant#, RegNum, Message)

The trigger application deals with the assignment of student grants for supporting teaching activities. Students applying for a student grant are inserted into a ranking (reported in table STUDENT RANKING). When a new grant becomes available, the student recipient of the grant is selected from the ranking. The same student may be the recipient of more than one grant, provided that she/he does not exceed 150 hours of teaching activities. Write the triggers managing the following tasks for the automatic assignment of student grants.

- (1) *Grant application.* A student applies for the assignment of a student grant (insertion into table GRANT\_APPLICATION). The application is accepted if (i) the student has acquired at least 120 credits on passed exams (i.e., on exams with score above 17) and (ii) the student is not yet in the ranking (i.e., in table STUDENT\_RANKING). If any of the two requirements is not satisfied, the application is rejected. If the application is accepted, the student is inserted in the ranking. The total points (attribute TotalPoints) of the student are given by the average score computed only on passed exams divided by the years elapsed from the student first enrollment (the current year is given by the variable YEAR(SYSDATE)).
- (2) When a new grant becomes available (insertion into table GRANT\_AVAILABILITY), the recipient student is selected from the ranking. The recipient is the student with the highest ranking that satisfies the following requirements: (i) she/he has passed the exam for the course on which the grant is available, and (ii) she/he does not exceed 150 teaching hours overall (including also the new grant). Suppose that at most one student satisfies the above requirements. If the grant is assigned, table GRANT\_ASSIGNMENT should be appropriately modified. The result of the assignment process must be notified both in the positive case (the grant is assigned) and in the negative case (no appropriate student is available, in this case the RegNum attribute takes the NULL value). The Not# attribute is a counter, which is incremented for each new notification.

## *Draft solution*

### Grant Application

```
CREATE OR REPLACE TRIGGER MANAGE_GRANT_APPLICATION
AFTER INSERT ON GRANT_APPLICATION
FOR EACH ROW
DECLARE
    TOTALCREDITS NUMBER;
    SCOREAVG NUMBER;
    X NUMBER;
    YEAR DATE;

BEGIN
    ---check if the student is in the ranking
    SELECT COUNT(*) INTO X
    FROM STUDENT_RANKING
    WHERE RegNum = :NEW.RegNum;

    -- check if the application can be accepted
    IF (X <> 0) THEN
        --- the application is rejected
        RAISE_APPLICATION_ERROR(-20500,'The application cannot be accepted');
    END IF;

    --- requirements verification
    SELECT SUM(Credits), AVG(Score) INTO TOTALCREDITS, SCOREAVG
    FROM EXAM_REGISTRATION E, COURSE C
    WHERE E.CourseCode = C.CourseCode
    AND RegNum = :NEW.RegNum AND Score ≥ 18;

    -- check if the application can be accepted
    IF (TOTALCREDITS<120) THEN
        --- the application is rejected
        RAISE_APPLICATION_ERROR(-20500,'The application cannot be accepted');
    END IF;

    ---the application is accepted
    ---insertion in the ranking
    SELECT YearFirstEnrollment INTO YEAR
    FROM STUDENT
    WHERE RegNum = :NEW.RegNum;

    INSERT INTO STUDENT_RANKING(RegNum, TotalPoints)
    VALUES (:NEW.RegNum, SCOREAVG/(YEAR(SYSDATE)- YEAR));

END;
```

## Grant Assignment

```
CREATE OR REPLACE TRIGGER GRANT_ASSIGNMENT
AFTER INSERT ON GRANT_AVAILABILITY
FOR EACH ROW
```

```
DECLARE
```

```
    X NUMBER;
```

```
    Y NUMBER;
```

```
    MYRegNum NUMBER;
```

```
BEGIN
```

```
--- check the existence of best student meeting constraints
```

```
---compute the maximum value of TotalPoints (if any)
```

```
SELECT MAX(TotalPoints) INTO X
```

```
FROM STUDENT_RANKING
```

```
WHERE RegNum IN
```

```
(SELECT RegNum
FROM EXAM_REGISTRATION
WHERE CourseCode = :NEW.CourseCode AND Score ≥ 18)
```

```
--- Students who have passed
the exam
```

```
AND RegNum NOT IN
```

```
(SELECT RegNum
FROM GRANT_ASSIGNMENT
GROUP BY RegNum
HAVING SUM(TeachingHours) + :NEW.TeachingHours >150);
```

```
-- Students who exceed 150
teaching hours
```

```
--- notification management
```

```
--- read the maximum value of NOT#
```

```
SELECT MAX(NOT#) INTO Y
```

```
FROM NOTIFICATION;
```

```
IF (Y IS NULL) THEN
```

```
    Y := 0;
```

```
END IF;
```

```
IF (X IS NOT NULL) THEN
```

```
    --- best student is assigned grant
```

```
    SELECT RegNum INTO MYRegNum
```

```
    FROM STUDENT_RANKING
```

```
    WHERE TotalPoints = X
```

```
    AND RegNum IN
```

```
        (SELECT RegNum
```

```
        FROM EXAM_REGISTRATION
```

```
        WHERE CourseCode = :NEW.CourseCode AND Score ≥ 18)
```

```
    AND RegNum NOT IN (SELECT RegNum
```

```
                        FROM GRANT_ASSIGNMENT
```

```
                        GROUP BY RegNum
```

```
                        HAVING SUM(TeachingHours) + :NEW.TeachingHours >150);
```

```
    INSERT INTO GRANT_ASSIGNMENT(Grant#, RegNum, TeachingHours)
```

```
VALUES (:NEW.Grant#,MYRegNum,:NEW.TeachingHours);  
INSERT INTO NOTIFICATION(NOT#, Grant#, RegNum, Message)  
VALUES (Y+1, :NEW.Grant#, MYRegNum, "GRANT ASSIGNED");
```

```
ELSE
```

```
--- no appropriate student found
```

```
INSERT INTO NOTIFICATION(NOT#, Grant#, RegNum, Message)  
VALUES (Y+1, :NEW.Grant#, NULL, "GRANT NOT ASSIGNED");
```

```
END IF;
```

```
END;
```