PL/SQL Repetition Exercise 1

2020-09-23

Instructions

- The tasks are based on the sample data used for the SQL Repetition Exercise.
- Create a package 'electronics_merchant' (header and body) which provides the functionalities listed below.
- **For each task also provide a sample (call) in the test driver!**
- Hand in header, body, test driver and the output of the test driver.
- **Important: if asked to modify something created in an earlier task you have to hand in the original and the new version!**

1 Employee Management

1.1 Paying a salary

The first step is to extend the model (this is not yet part of the package!). Add the following column to the *Employees* table:

- Salary, NUMBER(7,2), NOT NULL

Set the default value to 1 – we will fix the incorrect salary in the next step.

To assign each employee his/her correct salary we will need to consider the following:

- The base salary is 1750.
- Programmers (job title) get 250 extra.
- For every day someone worked at our company we increase the salary by 0.25.
- Some employees are managers, for each underling[1] we increase the salary by 50.

It is time to implement the first two methods of our PL/SQL package:

1.1.1 FUNCTION get_underling_count

- Counts all employees who have the supplied one as manager.
- Be careful: transitive management counts as well ($A \rightarrow B \;\&\; B \rightarrow C \Rightarrow A \rightarrow C$)
- Parameter:
  - The employee ID of the employee for whom we want to count his subordinates.
- Returns: the subordinate count

1.1.2 PROCEDURE set_salary

- Calculates the proper salary (rounded to two decimal places) for every employee.
- Updates the salary in the database (*use a cursor with write lock for this!*)
- Parameter: None

[1] Impolite name for an employee who works for (under) another one. Don't actually call someone that!

1.2 Adding a new employee

We need to hire new people, because thanks to our new and cutting edge PL/SQL management software we expect a revenue spike.

1.2.1 PROCEDURE add_employee

- Adds a new employee to the *Employees* table.
- Parameter:
  - A rt_new_employee record with information about the new employee.
- rt_new_employee fields:
  - first_name: The first name of the new employee.
  - last_name: The last name of the new employee.
  - email: The email of the new employee.
  - phone: The phone number of the new employee.
  - job_title: The job of the new employee.
- Be careful to only allow job titles which already exist in the company.
  - If an invalid job title is supplied, throw a proper exception with useful user information.
- Figure out a new employee_id for the new employee.

1.2.2 PROCEDURE set_manager

- Defines the manager for an employee
- Parameter:
  - Employee ID
  - Manager ID
- Make sure both employee and manager exist and are not the same person.
  - If not throw a proper exception as usual.

1.3 Salary TRIGGER

We now have two problems:

1. Our newly added employee does not have a salary (except of the minimal default one).
2. Salary is partly based on the number of people someone manages, so setting a manager changes these numbers.

The solution is quite simple: we need to recalculate salaries every time a new employee is added or a manager is set.

Instead of adding procedure calls at all kinds of places where we might update the employees table develop a trigger to automatically recalculate salaries when appropriate[2].

Mind, that a trigger is not bound to the package, so you have to create it globally and potentially deal with other code modifying the monitored tables as well – this won't happen in this case, but keep it in mind when using triggers.

[2]Important: under normal circumstances you would not update all salaries just because one or two rows changed. Instead you'd calculate and update the value just for the affected rows. To ease this process introducing a function which only calculates the salary without actually writing it would be beneficial since it could be used in both the main procedure and the trigger.

1.3.1 The easy way

Why won't this trigger work? And what condition is missing from it (based on the requirements described above)? Try it yourself and write down your findings.

```
create or replace trigger employee_salary
    after insert on employees
    begin
        if inserting then
            dbms_output.put_line('New employee added, calculating
                salaries');
        elsif updating then
            dbms_output.put_line('Manager set, calculating salaries'
                );
        end if;
        electronics_merchant.set_salary;
        dbms_output.put_line('Salaries updated');
    end;
```

1.3.2 The proper way

To get a version working without any tricks we will simply use a second table.

Create a table[3] *Compensations*[4] with the following columns:

- Employee_Id, , not null, primary key
  - Same data type as in *Employees* table
  - Primary key
  - Foreign key to *Employees* table
- Salary
  - Number(7,2)
  - NOT NULL
  - Default 1

Don't forget to create entries for all the existing employees.

Now you have four tasks:

1. Remove the now unnecessary *Salary* column from the *Employees* table.
2. Update your add_employee procedure to insert the proper entry in the new *Compensations* table.
3. Update your set_salary procedure to set the salary in the new table.
4. Finally create a trigger which fulfills the requirements described at the beginning.

[3]This is, of course, also not part of the package.

[4]Usually we do not use pluralized table names, but we make an exception here to stay in line with the sample data.