**Задание семинара 8**

**Лим В. БПИ225**

**1. Create Stored procedure**.

In this exercise, create a procedure to add a new job into the JOBS table.

a. Create a stored procedure called NEW\_JOB to enter a new order into the JOBS table. The procedure should accept three parameters. The first and second parameters supply a job ID and a job title. The third parameter supplies the minimum salary. Use the maximum salary for the new job as twice the minimum salary supplied for the job ID.

b. Invoke the procedure to add a new job with job ID 'SY\_ANAL', job title 'System Analyst', and minimum salary 6,000.

**Решение:**

--1.

CREATE OR REPLACE PROCEDURE *NEW\_JOB*(

p\_job\_id IN VARCHAR,

p\_job\_title IN VARCHAR,

p\_min\_salary IN INTEGER

)

LANGUAGE plpgsql

AS

$$

BEGIN

IF NOT *EXISTS* (SELECT 1 FROM jobs WHERE job\_id = p\_job\_id) THEN

INSERT INTO jobs (job\_id, job\_title, min\_salary, max\_salary)

VALUES (p\_job\_id, p\_job\_title, p\_min\_salary, p\_min\_salary \* 2);

ELSE

RAISE NOTICE 'Job ID % already exists.', p\_job\_id;

END IF;

END;

$$;

**2. Create Stored Procedure**.

In this exercise, create a program to add a new row to the JOB\_HISTORY table for an existing employee.

a. Create a stored procedure called ADD\_JOB\_HIST to add a new row into the JOB\_HISTORY table for an employee who is changing his job to the new job ID ('SY\_ANAL') that you created in exercise 1b.

The procedure should provide two parameters: one for the employee ID who is changing the job, and the second for the new job ID. Read the employee ID from the EMPLOYEES table and insert it into the JOB\_HISTORY table. Make the hire date of this employee as the start date and today’s date as the end date for this row in the JOB\_HISTORY table.

Change the hire date of this employee in the EMPLOYEES table to today’s date. Update the job ID of this employee to the job ID passed as parameter (use the 'SY\_ANAL' job ID) and salary equal to the minimum salary for that job ID plus 500.

Note: Include exception handling to handle an attempt to insert a nonexistent employee.

. Disable all triggers on the EMPLOYEES, JOBS, and JOB\_HISTORY tables before invoking the ADD\_JOB\_HIST procedure.

. Execute the procedure with employee ID 106 and job ID 'SY\_ANAL' as parameters.

. Query the JOB\_HISTORY and EMPLOYEES tables to view your changes for employee 106, and then commit the changes.

. Reenable the triggers on the EMPLOYEES, JOBS, and JOB\_HISTORY tables.

**Решение:**

--2.

CREATE OR REPLACE PROCEDURE *ADD\_JOB\_HIST*(

p\_employee\_id INTEGER,

p\_new\_job\_id VARCHAR(10)

)

LANGUAGE plpgsql

AS $$

DECLARE

v\_min\_salary INTEGER;

BEGIN

SELECT min\_salary INTO v\_min\_salary

FROM jobs

WHERE job\_id = p\_new\_job\_id;

INSERT INTO job\_history (employee\_id, start\_date, end\_date, job\_id, department\_id)

VALUES (p\_employee\_id, (SELECT hire\_date FROM employees WHERE employee\_id = p\_employee\_id), *CURRENT\_DATE*, p\_new\_job\_id,

(SELECT department\_id FROM employees WHERE employee\_id = p\_employee\_id));

UPDATE employees

SET job\_id = p\_new\_job\_id,

salary = v\_min\_salary + 500,

hire\_date = *CURRENT\_DATE*

WHERE employee\_id = p\_employee\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE EXCEPTION 'Employee not found';

END;

$$;

CALL *ADD\_JOB\_HIST*(106, 'SY\_ANAL');

**3. Create Stored Procedure**. In this exercise, create a program to update the minimum and maximum salaries for a job in the JOBS table.

a. Create a stored procedure called UPD\_JOBSAL to update the minimum and maximum salaries for a specific job ID in the JOBS table. The procedure should provide three parameters: the job ID, a new minimum salary, and a new maximum salary. Add exception handling to account for an invalid job ID in the JOBS table. Raise an exception if the maximum salary supplied is less than the minimum salary. Provide a message that will be displayed if the row in the JOBS table is locked.

Hint: The resource locked/busy error number is –54.

b. Execute the UPD\_JOBSAL procedure by using a job ID of 'SY\_ANAL', a minimum salary of 7000, and a maximum salary of 140.

c. Disable triggers on the EMPLOYEES and JOBS tables.

d. Execute the UPD\_JOBSAL procedure using a job ID of 'SY\_ANAL', a minimum salary of 7000, and a maximum salary of 14000.

e. Query the JOBS table to view your changes, and then commit the changes.

. Enable the triggers on the EMPLOYEES and JOBS tables.

**Решение:**

--3.

CREATE OR REPLACE PROCEDURE *UPD\_JOBSAL*(

p\_job\_id VARCHAR(10),

p\_new\_min\_salary INTEGER,

p\_new\_max\_salary INTEGER

)

LANGUAGE plpgsql

AS $$

BEGIN

IF p\_new\_max\_salary < p\_new\_min\_salary THEN

RAISE EXCEPTION 'Maximum salary cannot be less than minimum salary';

END IF;

UPDATE jobs

SET min\_salary = p\_new\_min\_salary,

max\_salary = p\_new\_max\_salary

WHERE job\_id = p\_job\_id;

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'Error while updating the salary for job\_id: %', p\_job\_id;

END;

$$;

CALL *UPD\_JOBSAL*('SY\_ANAL', 7000, 14000);

**4. Create Stored Function**.  Create a subprogram to retrieve the number of years of service for a specific employee.

a. Create a stored function called GET\_YEARS\_SERVICE to retrieve the total number of years of service for a specific employee. The function should accept the employee ID as a parameter and return the number of years of service. Add error handling to account for an invalid employee ID.

b. Invoke the GET\_YEARS\_SERVICE function in a call to DBMS\_OUTPUT.PUT\_LINE for an employee with ID 999.

c. Display the number of years of service for employee 106 with

DBMS\_OUTPUT.PUT\_LINE invoking the GET\_YEARS\_SERVICE function.

d. Query the JOB\_HISTORY and EMPLOYEES tables for the specified employee to verify that the modifications are accurate. Note: The values represented in the results on this page may differ from those you get when you run these queries.

**Решение:**

**--4.**

**CREATE OR REPLACE FUNCTION *GET\_YEARS\_SERVICE*(p\_employee\_id INT)**

**RETURNS INT AS**

**$$**

**DECLARE**

**v\_years\_service INT;**

**BEGIN**

**SELECT *EXTRACT*(YEAR FROM *CURRENT\_DATE*) - *EXTRACT*(YEAR FROM hire\_date)**

**INTO v\_years\_service**

**FROM employees**

**WHERE employee\_id = p\_employee\_id;**

**IF v\_years\_service IS NULL THEN**

**RAISE EXCEPTION 'Employee with ID % does not exist', p\_employee\_id;**

**END IF;**

**RETURN v\_years\_service;**

**END;**

**$$ LANGUAGE plpgsql;**

**5. Create Stored Function**. In this exercise, create a program to retrieve the number of different jobs that an employee worked on during his or her service.

a. Create a stored function called GET\_JOB\_COUNT to retrieve the total number of different jobs on which an employee worked.

The function should accept the employee ID in a parameter, and return the number of different jobs that the employee worked on until now, including the present job. Add exception handling to account for an invalid employee ID. Hint: Use the distinct job IDs from the JOB\_HISTORY table, and exclude the current job ID, if it is one of the job IDs on which the employee has already worked. Write a UNION of two queries and count the rows retrieved into a PL/SQL table. Use a FETCH with BULK COLLECT INTO to obtain the unique jobs for the employee.

b. Invoke the function for an employee with ID 176.

**Решение:**

--5.

CREATE OR REPLACE FUNCTION *GET\_JOB\_COUNT*(p\_employee\_id INTEGER)

RETURNS INTEGER AS $$

DECLARE

v\_job\_count INTEGER;

BEGIN

SELECT *COUNT*(DISTINCT job\_id) INTO v\_job\_count

FROM job\_history

WHERE employee\_id = p\_employee\_id

UNION

SELECT *COUNT*(DISTINCT job\_id)

FROM employees

WHERE employee\_id = p\_employee\_id;

RETURN v\_job\_count;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE EXCEPTION 'Employee not found';

END;

$$ LANGUAGE plpgsql;

SELECT *GET\_JOB\_COUNT*(176);

**6. Create Trigger.**

In this exercise, create a trigger to ensure that the minimum and maximum salaries of a job are never modified such that the salary of an existing employee with that job ID is outside the new range specified for the job.

a. Create a trigger called CHECK\_SAL\_RANGE that is fired before every row that is updated in the MIN\_SALARY and MAX\_SALARY columns in the JOBS table. For any minimum or maximum salary value that is changed, check whether the salary of any existing employee with that job ID in the EMPLOYEES table falls within the new range of salaries specified for this job ID. Include exception handling to cover a salary range change that affects the record of any existing employee.

b. Test the trigger using the SY\_ANAL job, setting the new minimum salary to 5000 and the new maximum salary to 7000. Before you make the change, write a query to display the current salary range for the SY\_ANAL job ID, and another query to display the employee ID, last name, and salary for the same job ID. After the update, query the change (if any) to the JOBS table for the specified job ID.

c. Using the job SY\_ANAL, set the new minimum salary to 7000 and the new maximum salary to 18000. Explain the results.

**Решение:**

--6.

CREATE OR REPLACE FUNCTION *check\_salary\_range*()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.min\_salary > NEW.max\_salary THEN

RAISE EXCEPTION 'Minimum salary cannot be greater than maximum salary';

END IF;

IF *EXISTS* (SELECT 1 FROM employees WHERE job\_id = OLD.job\_id AND (salary < NEW.min\_salary OR salary > NEW.max\_salary)) THEN

RAISE EXCEPTION 'Employee salary falls outside new salary range';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER CHECK\_SAL\_RANGE

BEFORE UPDATE ON jobs

FOR EACH ROW

EXECUTE FUNCTION check\_salary\_range();

UPDATE jobs

SET min\_salary = 5000, max\_salary = 7000

WHERE job\_id = 'SY\_ANAL';