*Kyoto Computer Gakuin – Advanced Topics in Database Technologies Roberto Espinoza –* [*r\_espinoza@kcg.edu*](mailto:r_espinoza@kcg.edu)

Advanced Topics in Database Technologies

– Practical Assignment – Advanced and Procedural SQL –

**For all exercises below, use the COMPANY database provided in class.**

**Remember to consult the relational diagram to find the correct relationship between tables.**

1. As part of the holidays, your company wants to provide a bonus to the employees that live in Houston. However, the bonus will apply according to their current salaries. If the employee has a salary above the average of the employees from Houston, they will get 1000. If it is equal to the average, the bonus will be of 2000. Finally, if their salary is less than the average, they will get 3000 for the bonus. Write an SQL statement that complies with these considerations and shows the ssn, the full name of the employee (in only 1 column), their current salary, and the assigned bonus. (20 points)

* SELECT e.ssn, (e.fname || ' ' || e.lname) AS "Full Name", e.salary,

CASE

WHEN e.salary > (SELECT AVG(e2.salary) FROM employee e2 WHERE e2.address LIKE '%Houston%') THEN 1000

WHEN e.salary = (SELECT AVG(e2.salary) FROM employee e2 WHERE e2.address LIKE '%Houston%') THEN 2000

ELSE 3000

END AS "Bonus"

FROM employee e WHERE e.address LIKE '%Houston%';

1. Trying to increase the quality of support to the employees, the company wants to periodically check the averages salaries for each department, and to restrict the registration of new employees with a salary lower than the lower average from all the departments. In that regard, write the following SQL statements:
   1. A view that allows to see the department name and code, the number of employees per department, and the average salary of the employees for each department. (12 points)

* CREATE VIEW dept\_salaries AS SELECT d.dname, d.dnumber, COUNT(e.ssn) AS "Number of Employees", AVG(e.salary) AS "Average Salary"

FROM department d

LEFT JOIN employee e ON e.dno = d.dnumber

GROUP BY d.dnumber, d.dname;

* 1. An update to the table Employee that creates a constraint that restricts the salary following the previous considerations. (8 points)
* ALTER TABLE employee ADD COLUMN min\_salary DECIMAL(10) NOT NULL DEFAULT 0;

CREATE OR REPLACE FUNCTION update\_min\_salary()

RETURNS TRIGGER AS $$

BEGIN

NEW.min\_salary := (SELECT AVG(e.salary) FROM employee e);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_min\_salary\_trigger

AFTER INSERT OR UPDATE ON employee

FOR EACH ROW EXECUTE PROCEDURE update\_min\_salary();

ALTER TABLE employee ADD CONSTRAINT salary\_constraint CHECK (salary >= min\_salary);

1. As part of the policy changes of the company to not overburden the employees, it will no longer be allowed to assign an employee to work for more than 40 hours total in the different projects. To that end, write the following SQL statements:
   1. A function that calculates and returns the number of hours left for each employee to reach the 40- hour limit. (12 points)

CREATE OR REPLACE FUNCTION hours\_left (essn CHAR (20))

RETURNS DECIMAL (3,1) AS $$

BEGIN

RETURN 40 - (SELECT SUM(w.Hours) FROM WORKS\_ON w WHERE w.essn = hours\_left.essn);

END;

$$ LANGUAGE plpgsql;

* 1. An update to the table Works\_on that creates a constraint that validates that any assignation of hours does not surpass the threshold of the total hours for employees. (8 points)
* CREATE OR REPLACE FUNCTION hours\_check()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.Hours + (SELECT SUM(w.Hours) FROM WORKS\_ON w WHERE w.essn = NEW.essn AND (w.essn, w.Pno) != (NEW.essn, NEW.Pno)) > 40 THEN

RAISE EXCEPTION 'Total hours cannot exceed 40 for employee';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER hours\_check\_trigger BEFORE INSERT OR UPDATE OF Hours ON WORKS\_ON FOR EACH ROW EXECUTE PROCEDURE hours\_check();

1. Similarly, another new policy for better communication between departments will be that each department can only have a maximum amount of 4 employees at a time. With that in consideration, write a trigger that allows you to validate such limit in the different possible scenarios that affect the number of employees. (20 points)

* CREATE OR REPLACE FUNCTION employee\_limit()

RETURNS TRIGGER AS $$

BEGIN

-- Check if the number of employees in the department would exceed the limit after the insert or update

IF (SELECT COUNT(e.ssn) FROM employee e WHERE e.dno = NEW.dno) + 1 > 4 THEN RAISE EXCEPTION 'Department cannot have more than 4 employees';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER employee\_limit\_trigger AFTER INSERT OR UPDATE OF dno ON employee FOR EACH ROW EXECUTE PROCEDURE employee\_limit();

1. Finally, to present the updated status of the departments, with the numbers of employees and projects associated to them, write a procedure that list as messages the name of the department, its code number, its number of employees, and its number of projects, for all the projects available in the company. (20 points)

* CREATE OR REPLACE PROCEDURE dept\_status()

AS $$

DECLARE

d department%ROWTYPE;

BEGIN

FOR d IN SELECT \* FROM department

LOOP

RAISE NOTICE 'department Name: %', d.dname;

RAISE NOTICE 'department Number: %', d.dnumber;

RAISE NOTICE 'Number of employees: %', (SELECT COUNT(e.ssn) FROM employee e WHERE e.dno = d.dnumber);

RAISE NOTICE 'Number of projects: %', (SELECT COUNT(p.pnumber) FROM project p WHERE p.dnum = d.dnumber);

END LOOP;

END;

$$ LANGUAGE plpgsql;

1