**Database Systems Management**

**Project – Database Management**

Lappeenranta-Lahti University of Technology LUT

Software Engineering

Database Systems Management

Spring 2023

Jongeun Park

Table of Contents

[1. The changes i have done to the database 1](#_Toc134373914)

[1.1 Create four views to provide for the company superiors and management. 1](#_Toc134373915)

[1.1.1View 1: 1](#_Toc134373916)

[1.1.2View 2: 2](#_Toc134373917)

[1.1.3View 3: 2](#_Toc134373918)

[1.1.4View 4: 3](#_Toc134373919)

[1.2 Create three triggers for the database: 3](#_Toc134373920)

[1.2.1Trigger 1: 3](#_Toc134373921)

[1.2.2Trigger 2: 4](#_Toc134373922)

[1.2.3Trigger 3: 5](#_Toc134373923)

[1.3 Create three procedures for the database: 6](#_Toc134373924)

[1.3.1Procedure 1: 6](#_Toc134373925)

[1.3.2Procedure 2: 7](#_Toc134373926)

[1.3.3Procedure 3: 8](#_Toc134373927)

[1.4 Partition following tables to at least three partitions (excluding default partition): 9](#_Toc134373928)

[1.4.1Employee partition table by contract type 9](#_Toc134373929)

[1.4.2project table by commission\_percentage 10](#_Toc134373930)

[1.5 Create access rights: 10](#_Toc134373931)

[1.6 Do the following changes to the database: 11](#_Toc134373932)

[1.6.1Add zip\_code column to Geo\_location (you don't have to populate it with data) 11](#_Toc134373933)

[1.6.2Add a NOT NULL constraint to customer email and project start date 12](#_Toc134373934)

[1.6.3Add a check constraint to employee salary and make sure it is more than 1000. You may have to update the salary information to be able to add the constraint (unless you have already done so) 12](#_Toc134373935)

# The changes i have done to the database

## Create four views to provide for the company superiors and management.

-The views should contain information that is important for the management of the company

- Idea of views is to combine data from various tables to provide an easier access to the combined information.

- The view does not have to be the final query result (i.e. View is used for easier access and then queried again for more detailed information)

- Each view should join at least two tables (not including linking tables)

### View 1:

-Name: view\_employees\_salary\_benefit

-Description: This view will provide information about the employees who has a salary benefit skill and order by e\_id

-Columns: e\_id, emp\_name, email, contract\_type, contract\_start, contract\_end, skill, salary\_benefit\_value

-Code:

Create view view\_employees\_salary\_benefit as

select e.e\_id, e.emp\_name, e.email, e.contract\_type, e.contract\_start, e.contract\_end, s.skill, s.salary\_benefit\_value

from employee e

join employee\_skills es on e.e\_id = es.e\_id

join skills s on es.s\_id = s.s\_id

where s.salary\_benefit = true

order by e.e\_id;

### View 2:

-Name: view\_employees\_has\_roject

-Description: This view will provide information about the employees who has a project and order by e\_id

-Columns: e\_id, emp\_name, email, contract\_type, contract\_start, contract\_end, project\_name, budget, commision\_percentage, p\_start\_date, p\_end\_date, c\_id

-Code:

Create view view\_employees\_has\_project as

select e.e\_id, e.emp\_name, e.email, e.contract\_type, e.contract\_start, e.contract\_end, p.project\_name, p.budget, p.commision\_percentage, p.p\_start\_date, p.p\_end\_date, p.c\_id

from employee e

join project\_role pr on e.e\_id = pr.e\_id

join project p on pr.p\_id = p.p\_id

order by e.e\_id;

### View 3:

-Name: view\_employees\_department\_hid1

-Description: This view will provide information about the employees who are in headquarter 1's departments(main office) and order by e\_id

-Columns: e\_id, emp\_name, email, contract\_type, contract\_start, contract\_end, department\_name, hq\_name

-Code:

Create view view\_employees\_department\_hid1 as

select e.e\_id, e.emp\_name, e.email, e.contract\_type, e.contract\_start, e.contract\_end, d.dep\_name, h.hq\_name

from employee e

join department d on e.d\_id = d.d\_id

join headquarters h on d.hid = h.h\_id

where h.h\_id = 1

order by e.e\_id;

### View 4:

-Name: view\_customers\_finland

-Description: This view will provide information about the customers who are from Finland and order by c\_id

-Columns: c\_id, c\_name, phone, email, street, city, country

-Code:

Create view view\_customers\_finland as

select c.c\_id, c.c\_name, c.phone, c.email, g.street, g.city, g.country

from customer c join geo\_location g on c.l\_id = g.l\_id

where g.country = 'Finland'

order by c.c\_id;

## Create three triggers for the database:

- One for before inserting a new skill, make sure that the same skill does not already exist

- One for after inserting a new project, check the customer country and select three employees from that country to start working with the project (i.e. create new project roles)

- One for before updating the employee contract type, make sure that the contract start date is also set to the current date and end date is either 2 years after the start date if contract is of Temporary type, NULL otherwise. (Temporary contract in Finnish is "määräaikainen". It's a contract that has an end date specified).

### Trigger 1:

-Name: trigger\_skill\_check

-Description: This trigger will check if the skill already exists in the database before inserting a new skill

-Code:

CREATE OR REPLACE FUNCTION trigger\_skill\_check()

RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT \* FROM skills WHERE skill = NEW.skill) THEN

RAISE EXCEPTION 'Skill already exists';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_duplicate\_skills

BEFORE INSERT ON skills

FOR EACH ROW

EXECUTE FUNCTION trigger\_skill\_check();

### Trigger 2:

-Name: trigger\_project\_role

-Description: This trigger will check the customer country and select three employees from that country to start working with the project (i.e. create new project roles)

-Code:

CREATE OR REPLACE FUNCTION trigger\_project\_role()

RETURNS TRIGGER AS $$

DECLARE

c\_country varchar(50);

BEGIN

SELECT l.country INTO c\_country FROM customer c JOIN geo\_location l ON c.l\_id = l.l\_id WHERE c.c\_id = NEW.c\_id;

INSERT INTO project\_role (e\_id, p\_id,prole\_start\_date)

SELECT e.e\_id, NEW.p\_id, NEW.p\_start\_date FROM employee e JOIN department d ON e.d\_id=d.d\_id

JOIN headquarters h ON d.hid=h.h\_id JOIN geo\_location gl ON h.l\_id=gl.l\_id WHERE gl.country = c\_country LIMIT 3;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER project\_role

AFTER INSERT ON project

FOR EACH ROW

EXECUTE FUNCTION trigger\_project\_role();

### Trigger 3:

-Name: trigger\_contract\_type

-Description: One for before updating the employee contract type, make sure that the contract start date is also set to the current date and end date is either 2 years after the start date if contract is of Temporary type, NULL otherwise.

-Code:

CREATE OR REPLACE FUNCTION trigger\_contract\_type()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.contract\_type = 'määräaikainen' OR NEW.contract\_type = 'Temporary' THEN

New.contract\_type = 'Temporary';

NEW.contract\_start = CURRENT\_DATE;

NEW.contract\_end = CURRENT\_DATE + INTERVAL '2 years';

ELSE

NEW.contract\_start = CURRENT\_DATE;

NEW.contract\_end = NULL;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER contract\_type

BEFORE UPDATE OF contract\_type ON employee

FOR EACH ROW

EXECUTE FUNCTION trigger\_contract\_type();

## Create three procedures for the database:

- Procedure that sets all employees salary to the base level based on their job title

- Procedure that adds 3 months to all temporary contracts (i.e: add 3 months to the contract end date)

- Procedure that increases salaries by a percentage based on the given percentage. You can also specify the highest salary to be increased (give limit X and salaries that are below X are increased).

- The user can specify the salary limit when calling the procedure. If user doesn't specify one (or gives 0 or null), then the limit is not considered. The percentage can be given in decimals or numbers or what ever you specify, as long as the procedure works.

### Procedure 1:

-Name: procedure\_set\_salary

-Description: This procedure will set all employees salary to the base level based on their job title table and I run this procedure.

-Code:

CREATE OR REPLACE PROCEDURE procedure\_set\_salary()

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE employee e

SET salary = jt.base\_salary

FROM job\_title jt

WHERE e.j\_id = jt.j\_id;

END;

$$;

CALL procedure\_set\_salary();

### Procedure 2:

-Name: procedure\_add\_months

-Description: This procedure will add 3 months to all temporary contracts (i.e: add 3 months to the contract end date)

-Code:

CREATE OR REPLACE PROCEDURE procedure\_add\_months()

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE employee e

SET contract\_end = contract\_end + INTERVAL '3 months'

WHERE e.contract\_type = 'Temporary';

END;

$$;

### Procedure 3:

-Name: procedure\_increase\_salary

-Description: This procedure will increase salaries by a percentage based on the given percentage. You can also specify the highest salary to be increased (give limit X and salaries that are below X are increased).

-Code:

CREATE OR REPLACE PROCEDURE procedure\_increase\_salary(

IN p\_percentage DECIMAL,

IN p\_limit INTEGER DEFAULT NULL

) AS $$

DECLARE

v\_employee\_row employee%ROWTYPE;

v\_new\_salary DECIMAL;

BEGIN

FOR v\_employee\_row IN SELECT \* FROM employee WHERE p\_limit IS NULL OR salary < p\_limit LOOP

v\_new\_salary := v\_employee\_row.salary \* (1 + p\_percentage/100);

IF p\_limit IS NOT NULL AND v\_new\_salary > p\_limit THEN

v\_new\_salary := p\_limit;

END IF;

UPDATE employee SET salary = v\_new\_salary WHERE e\_id = v\_employee\_row.e\_id;

END LOOP;

END;

$$ LANGUAGE plpgsql;

## Partition following tables to at least three partitions (excluding default partition):

- Note! You may have to create partitions based on the primary key unless you come up with another method

### Employee partition table by contract type

CREATE TABLE employee\_partitions (

e\_id integer NOT NULL,

emp\_name character varying COLLATE pg\_catalog."default" DEFAULT 'No Name'::character varying,

email character varying COLLATE pg\_catalog."default",

contract\_type character varying COLLATE pg\_catalog."default" NOT NULL,

contract\_start date NOT NULL,

contract\_end date,

salary integer DEFAULT 0,

supervisor integer,

d\_id integer,

j\_id integer

) PARTITION BY LIST (contract\_type);

CREATE TABLE employee\_parttime PARTITION OF employee\_partitions FOR VALUES IN ('Part-time');

CREATE TABLE employee\_temporary PARTITION OF employee\_partitions FOR VALUES IN ('Temporary');

CREATE TABLE employee\_fulltime PARTITION OF employee\_partitions FOR VALUES IN ('Full-time');

### project table by commission\_percentage

CREATE TABLE project\_partitions (

p\_id integer NOT NULL,

project\_name character varying COLLATE pg\_catalog."default",

budget numeric,

commission\_percentage numeric,

p\_start\_date date,

p\_end\_date date,

c\_id integer

) PARTITION BY RANGE (commission\_percentage);

CREATE TABLE project\_low PARTITION OF project\_partitions FOR VALUES FROM (0) TO (10);

CREATE TABLE project\_medium PARTITION OF project\_partitions FOR VALUES FROM (10) TO (20);

CREATE TABLE project\_high PARTITION OF project\_partitions FOR VALUES FROM (20) TO (100);

## Create access rights:

- Create three roles - admin, employee, trainee.

- Give admin all administrative rights (same rights as postgres superuser would have)

- Give employee rights to read all information but no rights to write

- Give trainee rights to read ONLY project, customer, geo\_location, and project\_role tables as well as limited access to employee table (only allow reading employee id, name, email)

CREATE ROLE admin;

CREATE ROLE employee;

CREATE ROLE trainee;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin;

GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO admin;

GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO admin;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO employee;

GRANT SELECT (e\_id, emp\_name, email) ON TABLE employee TO trainee;

GRANT SELECT ON TABLE project TO trainee;

GRANT SELECT ON TABLE customer TO trainee;

GRANT SELECT ON TABLE geo\_location TO trainee;

GRANT SELECT ON TABLE project\_role TO trainee;

## Do the following changes to the database:

### Add zip\_code column to Geo\_location (you don't have to populate it with data)

ALTER TABLE geo\_location ADD COLUMN zip\_code VARCHAR(5);

### Add a NOT NULL constraint to customer email and project start date

ALTER TABLE customer ALTER COLUMN email SET NOT NULL;

ALTER TABLE project ALTER COLUMN p\_start\_date SET NOT NULL;

### Add a check constraint to employee salary and make sure it is more than 1000. You may have to update the salary information to be able to add the constraint (unless you have already done so)

ALTER TABLE employee ALTER COLUMN salary SET NOT NULL;

ALTER TABLE employee ADD CONSTRAINT salary\_check CHECK (salary > 1000);