**Database Systems Management**

**Final Project**

**Sepideh Soleimani**

Lappeenranta-Lahti University of Technology LUT

Software Engineering

Spring 2024

TABLE OF CONTENTS

TABLE OF CONTENTS........................................................................................................ 1

1 VIEWS………..................................................................................................................... 2

2 TRIGGERS…...................................................................................................................... 4

3 PROCEDURES.................................................................................................................... 5

4 PARTITIONS....................................................................................................................... 6

5 ACCESS RIGHTS................................................................................................................ 7

6 CHANGES............................................................................................................................ 7

This report outlines the modifications and enhancements made to the database schema, including the creation of views, triggers, procedures, partitions, role management, and schema changes, aimed at improving data organization, access control, and performance.

**Task 1 (Views)**

The views should offer essential insights for company management, consolidating data from multiple tables to facilitate access to comprehensive information. They serve as an intermediary step for easier data retrieval and can be further queried for more detailed analysis. Each view should involve the combination of data from at least two tables, excluding linking tables.

* **Employee Performance Overview:**
  + Created a view named employee\_performance\_overview to provide a comprehensive overview of employees, including their IDs, names, emails, contract types, contract start and end dates, salaries, department names, and job titles.
  + Utilized JOIN operations with the employee, department, and job\_title tables to gather the necessary information.

CREATE OR REPLACE VIEW employee\_performance\_overview AS

SELECT

e.e\_id AS employee\_id,

e.emp\_name AS employee\_name,

e.email AS employee\_email,

e.contract\_type AS contract\_type,

e.contract\_start AS contract\_start\_date,

e.contract\_end AS contract\_end\_date,

e.salary AS salary,

d.dep\_name AS department\_name,

j.title AS job\_title

FROM

employee e

JOIN

department d ON e.d\_id = d.d\_id

JOIN

job\_title j ON e.j\_id = j.j\_id;

* **Project Performance Overview:**
  + Developed a view named project\_performance\_overview to present an overview of projects, including project IDs, names, start and end dates, customer names, and assigned employees.
  + Incorporated LEFT JOIN operations with the customer, project\_role, and employee tables to handle projects with no assigned employees.

CREATE OR REPLACE VIEW project\_performance\_overview AS

SELECT

p.p\_id AS project\_id,

p.project\_name AS project\_name,

p.p\_start\_date AS project\_start\_date,

p.p\_end\_date AS project\_end\_date,

c.c\_name AS customer\_name,

e.emp\_name AS assigned\_employee

FROM

project p

JOIN

customer c ON p.c\_id = c.c\_id

LEFT JOIN

project\_role pr ON p.p\_id = pr.p\_id

LEFT JOIN

employee e ON pr.e\_id = e.e\_id;

* **Department Employee Count:**
  + Constructed a view named department\_employee\_count to display the count of employees in each department.
  + Employed LEFT JOIN operation with the department and employee tables and utilized GROUP BY clause to aggregate the data.

CREATE OR REPLACE VIEW department\_employee\_count AS

SELECT

d.dep\_name AS department\_name,

COUNT(e.e\_id) AS employee\_count

FROM

department d

LEFT JOIN

employee e ON d.d\_id = e.d\_id

GROUP BY

d.dep\_name;

* **Skill Distribution:**
  + Established a view named skill\_distribution to demonstrate the distribution of skills among employees.
  + Utilized JOIN operation with the employee\_skills and skills tables and employed GROUP BY clause to aggregate the data.

CREATE OR REPLACE VIEW skill\_distribution AS

SELECT

s.skill AS skill\_name,

COUNT(es.e\_id) AS employee\_count

FROM

employee\_skills es

JOIN

skills s ON es.s\_id = s.s\_id

GROUP BY

s.skill;

**Task 2 (Triggers)**

* **Trigger One: Check Unique Skill:**
  + Implemented a trigger named before\_insert\_skill to ensure the uniqueness of skills before insertion into the skills table.
  + Utilized a PL/pgSQL function check\_unique\_skill() to validate if the skill being inserted already exists in the table.

DROP TRIGGER IF EXISTS before\_insert\_skill ON skills;

CREATE OR REPLACE FUNCTION check\_unique\_skill() RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT 1 FROM skills WHERE skill = NEW.skill) THEN

RAISE EXCEPTION 'Skill % already exists', NEW.skill;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_insert\_skill

BEFORE INSERT ON skills

FOR EACH ROW

EXECUTE FUNCTION check\_unique\_skill();

* **Trigger Two: Assign Employees to New Project:**
  + Developed a trigger named after\_insert\_project to automatically assign employees to a newly created project based on the country of the associated customer.
  + Utilized a PL/pgSQL function assign\_employees\_to\_project() to select three employees from the same country as the customer and insert project roles for them.

DROP TRIGGER IF EXISTS after\_insert\_project ON project;

CREATE TRIGGER after\_insert\_project

AFTER INSERT ON project

FOR EACH ROW

EXECUTE FUNCTION assign\_employees\_to\_project();

* **Trigger Three: Validate Employee Contract Update:**
  + Created a trigger named before\_update\_employee\_contract to validate and update employee contract details.
  + Utilized a PL/pgSQL function validate\_contract\_update() to adjust contract start and end dates based on the contract type.

CREATE OR REPLACE FUNCTION validate\_contract\_update() RETURNS TRIGGER AS $$

BEGIN

IF NEW.contract\_type = 'määräaikainen' THEN

NEW.contract\_start := CURRENT\_DATE;

NEW.contract\_end := NEW.contract\_start + INTERVAL '2 years';

ELSE

NEW.contract\_start := CURRENT\_DATE;

NEW.contract\_end := NULL;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_update\_employee\_contract

BEFORE UPDATE ON employee

FOR EACH ROW

WHEN (NEW.contract\_type <> OLD.contract\_type)

EXECUTE FUNCTION validate\_contract\_update();

**Task 3 (Procedures)**

* **Procedure One: Set Salary to Base Level Based on Job Title:**
  + Created a procedure named set\_salary\_to\_base\_level() to set employee salaries to the base level defined by their job titles.

CREATE OR REPLACE PROCEDURE set\_salary\_to\_base\_level() AS $$

BEGIN

UPDATE employee e

SET salary = (SELECT base\_salary FROM job\_title WHERE j\_id = e.j\_id);

END;

$$ LANGUAGE plpgsql;

* **Procedure Two: Add 3 Months to Temporary Contracts:**
  + Developed a procedure named extend\_temporary\_contracts() to extend the end date of temporary contracts by three months.

CREATE OR REPLACE PROCEDURE extend\_temporary\_contracts() AS $$

BEGIN

UPDATE employee

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

WHERE contract\_type = 'määräaikainen';

END;

$$ LANGUAGE plpgsql;

* **Procedure Three: Increase Salaries by a Percentage with Optional Salary Limit:**
  + Established a procedure named increase\_salaries() to increase employee salaries by a specified percentage, with an option to set a salary limit.

Limit

CREATE OR REPLACE PROCEDURE increase\_salaries(

IN percentage DECIMAL)

AS $$

DECLARE

v\_employee\_row employee%ROWTYPE;

v\_new\_salary DECIMAL;

BEGIN

FOR v\_employee\_row IN SELECT \* FROM employee LOOP

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

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

END LOOP;

END;

$$ LANGUAGE plpgsql;

**Task 4 (Partitions)**

* **Partitioning Employee and Customer Tables:**
  + Partitioned the employee and customer tables by e\_id and c\_id, respectively, using hash partitioning for better performance and manageability.

CREATE TABLE employee\_partitioned (

LIKE employee INCLUDING ALL

) PARTITION BY HASH (e\_id);

-- Create partitions

CREATE TABLE employee\_partition\_1 PARTITION OF employee\_partitioned FOR VALUES WITH (MODULUS 3, REMAINDER 0);

CREATE TABLE employee\_partition\_2 PARTITION OF employee\_partitioned FOR VALUES WITH (MODULUS 3, REMAINDER 1);

CREATE TABLE employee\_partition\_3 PARTITION OF employee\_partitioned FOR VALUES WITH (MODULUS 3, REMAINDER 2);

-- Partitioning the Customer table by c\_id

CREATE TABLE customer\_partitioned (

LIKE customer INCLUDING ALL

) PARTITION BY HASH (c\_id);

-- Create partitions

CREATE TABLE customer\_partition\_1 PARTITION OF customer\_partitioned FOR VALUES WITH (MODULUS 3, REMAINDER 0);

CREATE TABLE customer\_partition\_2 PARTITION OF customer\_partitioned FOR VALUES WITH (MODULUS 3, REMAINDER 1);

CREATE TABLE customer\_partition\_3 PARTITION OF customer\_partitioned FOR VALUES WITH (MODULUS 3, REMAINDER 2);

* **Partitioning Project Table:**
  + Partitioned the project table by commission\_percentage to optimize data retrieval and storage based on commission ranges.

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);

**Task 5 (Access Rights)**

* **Creating Roles:**
  + Created roles named admin and employee to manage access control within the database.

DO

$$

BEGIN

IF NOT EXISTS (SELECT FROM pg\_roles WHERE rolname = 'admin') THEN

CREATE ROLE admin WITH SUPERUSER;

END IF;

END

$$;

DO

$$

BEGIN

IF NOT EXISTS (SELECT FROM pg\_roles WHERE rolname = 'employee') THEN

CREATE ROLE employee;

END IF;

END

$$;

**Task 6 (Changes)**

* **Database Schema Changes:**
  + Added a zip\_code column to the Geo\_location table to enhance geographical data.
  + Set NOT NULL constraints on the email column of the Customer table and the start\_date column of the Project table to ensure data integrity.
  + Added a start\_date column to the Project table and populated it with current date values where null.
  + Updated salary information in the Employee table to ensure all salaries are above 1000 and added a check constraint to enforce this rule.

ALTER TABLE Geo\_location

ADD zip\_code VARCHAR(10);

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

ALTER TABLE Customer

ALTER COLUMN email SET NOT NULL;

-- Add start\_date column to Project table

ALTER TABLE Project

ADD start\_date DATE;

-- Update null start\_date values in Project table

UPDATE Project

SET start\_date = CURRENT\_DATE

WHERE start\_date IS NULL;

-- Set NOT NULL constraint on start\_date column

ALTER TABLE Project

ALTER COLUMN start\_date SET NOT NULL;

-- Update salary information to ensure all salaries are above 1000

UPDATE Employee

SET salary = 1200

WHERE salary < 1000;

-- Add a check constraint to employee salary to ensure it is more than 1000

ALTER TABLE Employee

ADD CONSTRAINT chk\_salary CHECK (salary > 1000);