DATABASE SYSTEM MANAGEMENT PROJECT REPORT

Aaron Mäkinen

Four Views 8 %

```
CREATE VIEW EmplyeeOverview AS

SELECT
    e.e_id,
    e.emp_name,
    e.email,
    t.title AS job_title,
    d.dep_name,
    e.salary,
    e.contract_type,
    e.contract_start,
    e.contract_end
FROM employee e
JOIN job_title t ON e.j_id = t.j_id
JOIN department d ON e.d_id = d.d_id;
```

– this will create employee over view where you can see all the details of the employee and it will join job_title and department tables on the right id.
– This can be called SELECT * FROM EmplyeeOverview
—----------

```
CREATE VIEW EmployeeProjectAssingments AS
        Select
    p.p_id,
    p.project_name,
    e.e_id,
    e.emp_name,
        e.contract_type,
        jt.title as job_title,
    pr.prole_start_date AS project_role
FROM project p
JOIN project_role pr ON p.p_id = pr.p_id
JOIN job_title jt ON p.p_id = jt.j_id
JOIN employee e ON pr.e_id = e.e_id;
```

– Here we have employeeProjectAssingments where we can see in which project employees are in. Also this uses four different tables.
– SELECT * FROM EmployeeProjectAssingments

```
CREATE VIEW EmployeeSkills AS

SELECT
        e.e_id,
        e.emp_name,
        es.s_id,
```

```
    s.skill


FROM employee e
JOIN employee_skills es ON e.e_id = es.e_id
JOIN skills s ON es.s_id = s.s_id;

– Here we can see each employes name and their skills
– Can be seen with SELECT * FROM EmployeeSkills


—----------------------------------------------

CREATE VIEW EmployeeUserGroup AS

SELECT
    e.e_id,
    e.emp_name,
    eug.u_id,
    ug.group_title,
    ug.group_rights


FROM employee e
JOIN employee_user_group eug ON e.e_id = eug.e_id
JOIN user_group ug ON eug.u_id = ug.u_id;


– Here we have employeeUserGroup that uses three tables. It will show employer name group_title and their rights.
– Can be seen with SELECT * EmployeeUserGroup
```

Trigger 15%

```
CREATE OR REPLACE FUNCTION reportSkillDublicate()
RETURNS TRIGGER
AS $$

BEGIN
	IF EXISTS(
		SELECT 1 FROM skills WHERE LOWER(skill) = LOWER(NEW.skill)
	) THEN
		RAISE EXCEPTION 'Given skill named "%" already exists', NEW.skill;
	END iF;
	RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER Skill_dublicate
BEFORE INSERT ON skills
FOR EACH ROW EXECUTE FUNCTION reportSkillDublicate();
```

–it will add a new programming language / skill if it is not in the database yet
–If the skill is there it will raise exception. This also notices if the language is given Java or java.

```
CREATE OR REPLACE FUNCTION assign_employees_to_project()
RETURNS TRIGGER AS $$
DECLARE
   customer_country TEXT;
   x RECORD;
BEGIN
   SELECT g.country INTO customer_country
   FROM customer c
   JOIN geo_location g ON c.l_id = g.l_id
   WHERE c.c_id = NEW.c_id;

   FOR x IN
      SELECT e.e_id
      FROM employee e
      JOIN department d ON e.d_id = d.d_id
      JOIN headquarters hq ON d.hid = hq.h_id
      JOIN geo_location g ON hq.l_id = g.l_id
      WHERE g.country = customer_country
      LIMIT 3
   LOOP
      INSERT INTO project_role (p_id, e_id, prole_start_date)
      VALUES (NEW.p_id, x.e_id, CURRENT_DATE);
```

```
    END LOOP;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS assign_employees_on_project ON project;

CREATE TRIGGER assign_employees_on_project
AFTER INSERT ON project
FOR EACH ROW
EXECUTE FUNCTION assign_employees_to_project();
```

–If inserting a new project, it will check the customer country and select three employees from that country to start working with the project

---

```
CREATE OR REPLACE FUNCTION update_contract_dates()
RETURNS TRIGGER AS $$
BEGIN

    NEW.contract_start := CURRENT_DATE;

    IF NEW.contract_type = 'Temporary' THEN
        NEW.contract_end := CURRENT_DATE + INTERVAL '2 years';
    ELSE
        NEW.contract_end := NULL;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER update_contract_dates
BEFORE UPDATE OF contract_type ON employee
FOR EACH ROW
EXECUTE FUNCTION update_contract_dates();
```

– **This updates contract type on employee. It will check if the contract type is temporary and if it is it will make a new contract end adding two years to the current date. Else it will set it to NULL**

Procedures 9%

```
CREATE OR REPLACE PROCEDURE set_salary()
LANGUAGE plpgsql
AS $$
BEGIN
	UPDATE employee s
		SET salary = jt.base_salary
		FROM job_title jt
		WHERE s.j_id = jt.j_id;
END;
$$;

--CALL set_salary();
```

–In this we set the salary based on base_salary from job_title to make sure it gives right salaries to each employee we make sure of it making a condition s.j_id = jt_id
–When using call set_salary it will update the right salaries

---

```
CREATE OR REPLACE PROCEDURE add_3_months_temporary_contracts()
LANGUAGE plpgsql
AS $$
BEGIN
	UPDATE employee
		SET contract_end = contract_end + INTERVAL '3 months'
		WHERE contract_type = 'Temporary';
END;
$$;

--CALL add_3_months_temporary_contracts();
```

– In this procedure we update the contract end to gice it three more months to contract_types that has Temporary in it.

```
CREATE OR REPLACE PROCEDURE increase_salaries(
	percentage NUMERIC,
	salarycap INTEGER DEFAULT 0

)
```

```
LANGUAGE plpgsql
AS $$
BEGIN
        IF salarycap IS NULL OR salarycap = 0 THEN
      UPDATE employee
                SET salary = salary + (salary * percentage / 100);
        ELSE
                UPDATE employee
                SET salary = salary + (salary * percentage / 100)
                WHERE salary < salarycap;
        END IF;

END;
$$;
--CALL increase_salaries(15, 5000);
```

–In this procedure we add percentage and salarycap. If user gives null or o, the limit is not considered.
–When calling this function it will update salaries under the salarycap that was given as a input into the procedure.

Partitions 8%

```
CREATE TABLE employee_partition (
   e_id INT NOT NULL,
   emp_name VARCHAR DEFAULT 'No Name',
        email VARCHAR,
   contract_type VARCHAR NOT NULL,
   contract_start DATE NOT NULL,
   contract_end DATE,
   salary INT DEFAULT 0,
   d_id INT,
   j_id INT,
   supervisor INT

) PARTITION BY RANGE (e_id);

CREATE TABLE employee_p1
        PARTITION OF employee_partition
        FOR VALUES FROM (0) TO (2000);

CREATE TABLE employee_p2
        PARTITION OF employee_partition
        FOR VALUES FROM (2000) TO (4000);

CREATE TABLE employee_p3
        PARTITION OF employee_partition
        FOR VALUES FROM (4000) TO (10000);

INSERT INTO employee_partition
SELECT * FROM employee;
```

--In these tables the partition divides e_id:s into its own sections. First table shows values from 0 to 2000 and second table (employee_p2) shows 2000 to 4000 and so on –to get the data we use command insert into employee_partition SELECT * FROM employee

---

-- **if inserting null for project p_star_date this this might not work, so when testin no nulls need to update (explained at last page)**

```
CREATE TABLE project_partition (
   p_id INT NOT NULL,
   project_name VARCHAR,
   budget NUMERIC,
   commission_percentage NUMERIC,
```

```
    p_start_date DATE,
    p_end_date DATE,
    c_id INT

) PARTITION BY RANGE (p_start_date);

CREATE TABLE start_date_p1
        PARTITION OF project_partition
        FOR VALUES FROM ('2000-01-01') TO ('2008-01-01');

CREATE TABLE start_date_p2
        PARTITION OF project_partition
        FOR VALUES FROM ('2008-01-01') TO ('2015-01-01');
CREATE TABLE start_date_p3
        PARTITION OF project_partition
        FOR VALUES FROM ('2015-01-01') TO ('2026-01-01');


INSERT INTO project_partition SELECT * FROM project;
```

--In these tables the partition divides p_start_date:s into its own sections. First table shows values from (‘2000-01-01’) to (‘2008-01-01’ and second table has its own timeframe and so on
– to get the data we use command insert into project_partition SELECT * FROM project

ROLES (6 %)

--1
CREATE ROLE admin WITH LOGIN; **-- Might be already**

CREATE ROLE employee WITH LOGIN; -- **Might be already**

CREATE ROLE trainee WITH LOGIN; -- **Might be already**

--2
ALTER ROLE admin WITH SUPERUSER; – **after this should work propely**

--3

GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO employee;
– **gave all read rights with select to empoloyee**

--4

GRANT SELECT ON public.project, public.customer, public.geo_location, public.project_role TO trainee; — **gave right to read only these tables**

GRANT SELECT (e_id, emp_name, email) ON public.employee TO trainee;

– Gave limited acces to employee to trainee

Last changes 4 (%)

--1
ALTER TABLE geo_location
ADD COLUMN zip_code character varying;

– **altered to add zip_code**

--2

UPDATE project
SET p_start_date = CURRENT_DATE
WHERE p_start_date IS NULL;

— **updated project in a way that p_start_date is the current date. This should be done if project_partition partitions start_date_p1 start_date_p2 start_date_p3 are not working**

UPDATE customer
SET email = 'example@gmail.com'
WHERE email IS NULL;

– **updated email if there is nulls**

ALTER TABLE customer
ALTER COLUMN email SET NOT NULL;

– **alter the column**

ALTER TABLE project
ALTER COLUMN p_start_date SET NOT NULL;
– **alter the column**
--3

UPDATE employee SET salary = 1001 WHERE salary <= 1000;

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

– **Add check constraint to employee salary and made sure it is more than 1000**