EPIC REPORT

Contents

Roles:

Made roles:

- Admin
- Employee
- Trainee
- Views_only

We gave admin superuser status.

We gave employee predefined role ‘pg_read_all_data’ which gives: “Read all data (tables, views, sequences), as if having SELECT rights on those objects, and USAGE rights on all schemas, even without having it explicitly. This role does not have the role attribute BYPASSRLS set. If RLS is being used, an administrator may wish to set BYPASSRLS on roles which this role is GRANTed to.” – postgresql documentation, e-documentation, referenced [21/04/2024], available at https://www.postgresql.org/docs/current/predefined-roles.html

Trainee has been granted ability to read tables project, customer, geo_location, and project_role. It also has permission to see traineeView which consists information from employee table rows id, name and email.

Views_only has the permission to select on the created views.

Done by running:

```
CREATE ROLE admin SUPERUSER;

CREATE ROLE employee;

CREATE ROLE trainee;

CREATE ROLE views_only;


GRANT pg_read_all_data TO employee;

GRANT SELECT ON project, customer, geo_location, project_role TO trainee;


create or replace view traineeView

as select e_id, emp_name, email from employee;


GRANT SELECT ON traineeView TO trainee;


GRANT SELECT ON employees_on_project, employees_in_department_in_hq,
employees_by_skill, customers_by_location, employees_by_group, employees_by_title TO
views_only;
```

Triggers

Made triggers:

- skillCheck
- assignTrigger
- contractTrigger
- groupTrigger

With each having their own procedure

- skillChecking
- assignEmployees
- contractCheck
- groupCheck

Each trigger only calls the corresponding procedure. Here's table to show when they trigger and what is the corresponding procedure

Trigger	Procedure	when
skillCheck	skillCheck	before insert on skills
assignTrigger	assignTrigger	after insert on project
contractTrigger	contractTrigger	before update of contract_type on employee
groupTrigger	groupCheck	After insert on employee

Explanation what each trigger does:

skillCheck:

Checks that is there skill with same name than the new input.

assignTrigger:

finds all employees that are in same country than the customer and chooses 3 on the top found to be in the project.

contractTrigger

checks that contract start date is today. Checks that there is an end date for temporary contract and that it is exactly two years after current day. Checks that there is no end date for non temporary contracts.

groupTrigger

checks if given job title 'HR secretary' and insert that employee to HR group. Checks if job title contains world admin and inserts that into admin group. Else they go employee group.

Done by running:

```
Create or replace function skillChecking() returns trigger
LANGUAGE plpgsql
AS $$
BEGIN

IF (SELECT count(*) FROM skills where skills.skill = new.skill) > 0 THEN RAISE EXCEPTION 'This skill already exists';
END IF;

RETURN NEW;
END;
$$;

Create or replace trigger skillCheck
before insert on skills
FOR EACH ROW
EXECUTE function skillChecking();

Create or replace function assignEmployees() returns trigger
LANGUAGE plpgsql
AS $$

DECLARE

    employee1 integer := (select e_id from employee where employee.e_id in
                        (select d_id from department where department.hid in
                         (select h_id from headquarters where headquarters.l_id in
                        (select l_id from geo_location where geo_location.l_id > 999 and
                        geo_location.country
```

```
                                    in (select country from geo_location where
                                    geo_location.l_id = (select l_id from customer where
                                    customer.c_id = 5))))) limit 1 offset 0);
    employee2 integer := (select e_id from employee where employee.e_id in
                                     (select d_id from department where department.hid in
                                     (select h_id from headquarters where headquarters.l_id in
                                    (select l_id from geo_location where geo_location.l_id > 999 and
                                    geo_location.country
                                     in (select country from geo_location where geo_location.l_id =
                                    (select l_id from customer where customer.c_id = 5))))) limit 1
                                    offset 1);
    employee3 integer := (select e_id from employee where employee.e_id in
                                     (select d_id from department where department.hid in
                                      (select h_id from headquarters where headquarters.l_id in
                                    (select l_id from geo_location where geo_location.l_id > 999 and
                                    geo_location.country
                                     in (select country from geo_location where geo_location.l_id =
                                    (select l_id from customer where customer.c_id = 5))))) limit 1
                                    offset 2);

BEGIN

INSERT INTO project_role (e_id, p_id, prole_start_date) VALUES(employee1, new.p_id, (SELECT
CURRENT_DATE));

INSERT INTO project_role (e_id, p_id, prole_start_date) VALUES(employee2, new.p_id, (SELECT
CURRENT_DATE));

INSERT INTO project_role (e_id, p_id, prole_start_date) VALUES(employee3, new.p_id, (SELECT
CURRENT_DATE));

--RAISE NOTICE ':D';

RETURN NEW;

END;

$$;


Create or replace trigger assignTrigger

after insert on project

FOR EACH ROW
```

```
EXECUTE function assignEmployees();



Create or replace function contractCheck() returns trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF ((SELECT CURRENT_DATE) != new.contract_start) THEN RAISE EXCEPTION 'Contract is
invalid, Start date is not today';

ELSIF new.contract_type = 'Temporary' and ((SELECT CURRENT_DATE) + interval '2' year !=
new.contract_end or new.contract_end IS NULL) THEN RAISE EXCEPTION 'Contract is invalid,
Contract should end 2 years from now';

ELSIF new.contract_type != 'Temporary' and new.contract_end IS NOT NULL THEN RAISE
EXCEPTION 'Contract is invalid, Contract should not have end';

END IF;

RETURN NEW;

END;

$$;


Create or replace trigger contractTrigger

before update of contract_type on employee

FOR EACH ROW

EXECUTE function contractCheck();



Create or replace function groupCheck() returns trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF 'HR secretary' = (SELECT title from job_title where job_title.j_id = new.j_id) THEN INSERT INTO
employee_user_group(e_id, u_id, eug_join_date) values(new.e_id, 6, (SELECT CURRENT_DATE));
-- u_id 6 = HR group
```

```
ELSIF (SELECT title from job_title where job_title.j_id = new.j_id) like '%admin%' THEN INSERT INTO employee_user_group(e_id, u_id, eug_join_date) values(new.e_id, 3, (SELECT CURRENT_DATE)); -- u_id 3 = admin group

ELSE INSERT INTO employee_user_group(e_id, u_id, eug_join_date) values(new.e_id, 9, (SELECT CURRENT_DATE)); -- u_id 9 = employee group

END IF;

RETURN NEW;

END;

$$;


Create or replace trigger groupTrigger

after insert on employee

FOR EACH ROW

EXECUTE function groupCheck();
```

Procedures

Made procedures:

- salaryBase()
- temporaryIncrease()
- percentSalaryIncrease(percentValue numeric, maximumValue numeric)
- correctSalary()

SalaryBase sets all employees salary to the salary given by their job title.

TemporaryIncrease gives all employees with temporary contract 3 months more contract time

percentSalaryIncrease takes in any numeric values. percentValue proceeds with integers being per cents like 20 = 20% and so on. Can use negative value. It also takes maximum value and if the original value was higher than the given value then the value wasn't increased. percentSalaryIncrease increased current salary value by given per cent value. If given null as salary increase nothing happens.

correctSalary first call salaryBase procedure to give them their salary a base value and then gives them additional salary for each benefit salary marked in the skills they have.

Done by running:

```
Create or replace procedure salaryBase()

LANGUAGE plpgsql

AS $$

BEGIN


UPDATE employee SET salary = (SELECT base_salary from job_title where job_title.j_id = employee.j_id);


END;

$$;
```

```
Create or replace procedure temporaryIncrease()

LANGUAGE plpgsql

AS $$

BEGIN


UPDATE employee SET contract_end = employee.contract_end + interval '3' month where employee.contract_type = 'Temporary';


END;

$$;
```

```
Create or replace procedure percentSalaryIncrease(percentValue numeric, maxiumValue numeric)

LANGUAGE plpgsql

AS $$

BEGIN


IF (maxiumValue IS NULL or maxiumValue = 0) and percentValue IS NOT NULL THEN UPDATE employee SET salary = salary*(1+(percentValue/100));
```

```
ELSIF percentValue IS NOT NULL THEN UPDATE employee SET salary =
salary*(1+(percentValue/100)) where employee.salary < maxiumValue;

END IF;


END;

$$;



Create or replace procedure correctSalary()

LANGUAGE plpgsql

AS $$

BEGIN

CALL salaryBase();

UPDATE employee SET salary = salary + (SELECT SUM(salary_benefit_value) FROM skills where
skills.s_id in (SELECT s_id from employee_skills where employee_skills.e_id = employee.e_id ));


END;

$$;
```

Functions

Made function:

- getProjects(givenDate date)

getProjects takes a date and returns in a table all projects which end date is later than given date and start date is earlier than given date. In the table are information about the project and the customer information.

Done by running:

```
Create or replace function getProjects(givenDate date) returns table(project_id int, name
varchar, budget numeric, commission_percentage numeric, start_date date, end_date date,
c_id int, customer_id int, c_name varchar, c_type varchar, phone varchar, email varchar, l_id int)

LANGUAGE plpgsql

AS $$

BEGIN


return query (SELECT * from project inner join customer on project.c_id = customer.c_id where
p_end_date > givenDate and p_start_date <= givenDate);


END;

$$;
```

Constraints

Added following constraints:

- *customer*: *email* not null
- *project*: *p_start_date* not null
- *employee*: *salary* > 1000

Done by running:

```
ALTER TABLE customer
ALTER COLUMN email SET NOT NULL;

ALTER TABLE project
ALTER COLUMN p_start_date SET NOT NULL;

-- change all salaries from 0 to 1001 to not get errors
UPDATE employee
SET salary = 1001
WHERE salary = 0;

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

Partitions

PostgreSQL does not support partitioning an existing table so instead the tables were copied except the copies were partitioned by the corresponding column and then renamed.

The following partitions were made this way:

- *customer* by *c_id*
- *project* by *p_id*

Each table has 3 partitions and a default partition.

After the tables have been renamed, all foreign keys referring to the original tables are updated to refer to the new ones. (project should have a fkey c_id but in the backup given, the constraint does not exist so it was created)

Done by running:

```
CREATE TABLE new_customer (LIKE customer INCLUDING ALL)
        PARTiTION BY RANGE (c_id);

CREATE TABLE customer_1 PARTITION OF new_customer
        FOR VALUES FROM (1) TO (300);

CREATE TABLE customer_2 PARTITION OF new_customer
        FOR VALUES FROM (300) TO (600);

CREATE TABLE customer_3 PARTITION OF new_customer
        FOR VALUES FROM (600) TO (900);

CREATE TABLE customer_default PARTITION OF new_customer
        DEFAULT;

INSERT INTO new_customer SELECT * FROM customer;

BEGIN;
ALTER TABLE customer RENAME TO old_customer;
ALTER TABLE new_customer RENAME TO customer;
COMMIT;

ALTER TABLE project add constraint project_c_id_fkey FOREIGN KEY (c_id) REFERENCES
customer(c_id);

CREATE TABLE new_project (LIKE project INCLUDING ALL)
        PARTiTION BY RANGE (p_id);

CREATE TABLE project_1 PARTITION OF new_project
        FOR VALUES FROM (1) TO (300);
```

```
CREATE TABLE project_2 PARTITION OF new_project
        FOR VALUES FROM (300) TO (600);

CREATE TABLE project_3 PARTITION OF new_project
        FOR VALUES FROM (600) TO (900);

CREATE TABLE project_default PARTITION OF new_project
        DEFAULT;

INSERT INTO new_project SELECT * FROM project;

BEGIN;
ALTER TABLE project RENAME TO old_project;
ALTER TABLE new_project RENAME TO project;
COMMIT;

ALTER TABLE project_role drop constraint project_role_p_id_fkey;
ALTER TABLE project_role add constraint project_role_p_id_fkey FOREIGN KEY (p_id)
REFERENCES project(p_id);
```

Views

Created the following views:

- employees_on_project
  lists all employees on a project

- employees_in_department_in_hq
  lists all employees by department and headquarter

- employees_by_skill
  lists all employees by their skills

- customers_by_location
  lists all customers by their location

- employees_by_group
  lists all employees by the group(s) they belong to

- employees_by_title
  lists all employees by the job title(s) they have

Done by running:

```
CREATE VIEW employees_on_project AS

SELECT p.project_name AS project_name, e.emp_name AS employee_name,

p.p_id AS p_id, e.e_id AS e_id

FROM project p, project_role r, employee e

WHERE r.e_id = e.e_id AND r.p_id = p.p_id

ORDER BY p.p_id, employee_name;


CREATE VIEW employees_in_department_in_hq AS

SELECT h.hq_name, d.dep_name, e.emp_name, j.title, e.salary, h.h_id AS h_id, d.d_id AS d_id,
e.e_id AS e_id

FROM headquarters h, department d, employee e, job_title j

WHERE e.d_id = d.d_id AND d.hid = h.h_id AND e.j_id = j.j_id

ORDER BY h.hq_name, d.dep_name, e.emp_name, j.title;


CREATE VIEW employees_by_skill AS

SELECT e.emp_name, s.skill, j.title, e.e_id AS e_id, s.s_id AS s_id

FROM employee AS e, employee_skills AS es, skills AS s, job_title AS j

WHERE e.e_id = es.e_id AND s.s_id = es.s_id AND e.j_id = j.j_id

ORDER BY e.emp_name, s.skill;


CREATE VIEW customers_by_location AS

SELECT c.c_name AS customer, g.country AS country, g.city AS city, g.street AS street, c.c_id AS
c_id, g.l_id AS l_id

FROM geo_location AS g, customer AS c

WHERE c.l_id = g.l_id

ORDER BY c.c_name, g.country, g.city, g.street;


CREATE VIEW employees_by_group AS

SELECT e.emp_name AS employee, g.group_title AS group, g.group_rights AS rights, e.e_id AS
e_id, g.u_id AS u_id

FROM employee AS e, employee_user_group AS eg, user_group AS g
```

```
WHERE e.e_id = eg.e_id AND g.u_id = eg.u_id

ORDER BY e.emp_name, g.group_title, g.group_rights;


CREATE VIEW employees_by_title AS

SELECT e.emp_name AS employee, j.title AS title, e.contract_type AS contract_type, e.salary AS
salary,

e.e_id AS e_id, j.j_id AS j_id

FROM employee e, job_title j

WHERE e.j_id = j.j_id

ORDER BY j.j_id, e.j_id;
```

Zip_code

Added a *zip_code* column to *geo_location*. It holds an integer and has no constraints.

Done by running:

ALTER TABLE geo_location ADD COLUMN zip_code INTEGER;