Университет ИТМО

Лабораторная работа № 5

«Процедуры, функции, триггеры в PostgreSQL»

Выполнила: Анисимова Ксения Сергеевна
Группа: K3241
Преподаватель: Говорова Марина Михайловна

Санкт – Петербург
2023

Цель работы: овладеть практическими навыками создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, SQL Shell (psql).

Практическое задание:

1) Создать хранимые процедуры:
   - Для получения расписания занятий для групп на определенный день недели.
   - Записи на курс слушателя.
   - Получения перечня свободных лекционных аудиторий на любой день недели. Если свободных аудиторий не имеется, то выдать соответствующее сообщение.

2) Создать триггер для логирования событий вставки, удаления, редактирования данных в таблице

Ход работы:

1) Процедура для получения расписания занятий для групп на определенный день недели:

```
create or replace function getscheduleforgroup(
    in group_name_param varchar(32),
    in week_day_param varchar(12)
)
returns table (
    class_id integer,
    discipline_name varchar(64),
    group_name varchar(32),
    facility_name varchar(32),
    address varchar(500),
    room_number varchar(32),
    week_day varchar(12),
    period integer
)
language plpgsql
as $$
begin
    return query
    select
        t.class_id,
        d.discipline_name,
        t.group_name,
        f.facility_name,
        f.address,
        c.room_number,
        t.week_day,
        t.period
    from
        courses.timetable t
        join courses.discipline d on d.discipline_id = t.discipline_id
        join courses.classroom c on t.classroom_id = c.classroom_id
        join courses.facility f on f.facility_id = c.facility_id
    where
        t.group_name = group_name_param
        and t.week_day = week_day_param;
end;
$$;
```

```
create or replace function getscheduleforgroup(
   in group_name_param varchar(32),
   in week_day_param varchar(12)
)
returns table (
   class_id integer,
   discipline_name varchar(64),
   group_name varchar(32),
   facility_name varchar(32),
   address varchar(500),
   room_number varchar(32),
   week_day varchar(12),
   period integer
)
language plpgsql
as $$
begin
   return query
   select
      t.class_id,
      d.discipline_name,
      t.group_name,
      f.facility_name,
      f.address,
      c.room_number,
      t.week_day,
      t.period
   from
      courses.timetable t
      join courses.discipline d on d.discipline_id = t.discipline_id
      join courses.classroom c on t.classroom_id = c.classroom_id
      join courses.facility f on f.facility_id = c.facility_id
   where
      t.group_name = group_name_param
      and t.week_day = week_day_param;
end;
$$;
```

```
COURSES=# SELECT * FROM GetScheduleForGroup('M11', 'ПН');
 class_id |      discipline_name       | group_name |     facility_name      |                address                 | room_number | week_day | period
----------+----------------------------+------------+------------------------+----------------------------------------+-------------+----------+--------
        1 | Алгебра. Начальный уровень | M11        | Корпус им. Солженицына | Россия, г. Воронеж, Юбилейная ул., д. 11 | j12         | ПН       |      1
        2 | Алгебра. Начальный уровень | M11        | Корпус им. Солженицына | Россия, г. Воронеж, Юбилейная ул., д. 11 | j12         | ПН       |      2
        3 | Алгебра. Начальный уровень | M11        | Корпус им. Солженицына | Россия, г. Воронеж, Юбилейная ул., д. 11 | j12         | ПН       |      3
(3 rows)
```

```
SELECT * FROM GetScheduleForGroup('M11', 'ПН');
```

2) Процедура для записи на курс слушателя:

```
create or replace procedure enrolllistener(
    in listener_passport_param bigint,
    in program_id_param integer
)
language plpgsql
as $$
begin

    if not exists (select 1 from courses.listener where passport = listener_passport_param) then
        raise exception 'Слушатель с паспортом % не найден', listener_passport_param;
    end if;

    if not exists (select 1 from courses.program where program_id = program_id_param) then
        raise exception 'Программа с ID % не найдена', program_id_param;
    end if;

    if exists (
        select 1
        from courses.education e
        join courses.group g on e.group_name = g.group_name
        where e.passport = listener_passport_param and g.program_id = program_id_param
    ) then
        raise exception 'Этот слушатель уже записан на эту программу';
    end if;

    insert into courses.education (listener_number, status, group_name, passport)
    values (
        (select (max(listener_number)+1) from courses.education ),
        'обучается',
        (select group_name from courses.group where program_id = program_id_param order by random() limit 1),
        listener_passport_param
    );
    raise notice 'Слушатель % записан в программу %', listener_passport_param, program_id_param;
end;
$$;
```

```
create or replace procedure enrolllistener(
  in listener_passport_param bigint,
  in program_id_param integer
)
language plpgsql
as $$
begin

  if not exists (select 1 from courses.listener where passport = listener_passport_param) then
    raise exception 'Слушатель с паспортом % не найден', listener_passport_param;
  end if;

  if not exists (select 1 from courses.program where program_id = program_id_param) then
    raise exception 'Программа с ID % не найдена', program_id_param;
  end if;

  if exists (
    select 1
    from courses.education e
    join courses.group g on e.group_name = g.group_name
    where e.passport = listener_passport_param and g.program_id = program_id_param
  ) then
    raise exception 'Этот слушатель уже записан на эту программу';
  end if;

  insert into courses.education (listener_number, status, group_name, passport)
  values (
    (select (max(listener_number)+1) from courses.education ),
    'обучается',
    (select group_name from courses.group where program_id = program_id_param order by random() limit
1),
```

```
        listener_passport_param
    );
    raise notice 'Слушатель % записан в программу %', listener_passport_param, program_id_param;
end;
$$;
```

```
COURSES=# CALL EnrollListener(5551123399, 118);
NOTICE:  Слушатель 5551123399 записан в программу 118
CALL
COURSES=# CALL EnrollListener(5551123399, 118);
ERROR:  Этот слушатель уже записан на эту программу
CONTEXT:  PL/pgSQL function enrolllistener(bigint,integer) line 18 at RAISE
```

CALL EnrollListener(5551123399, 118); (И повторно)

3) Процедура для получения перечня свободных лекционных аудиторий на любой день недели. Если свободных аудиторий не имеется, то выдать соответствующее сообщение:

```
CREATE OR REPLACE FUNCTION GetFreeClassrooms(
    IN week_day_param VARCHAR(12)
)
RETURNS TABLE (
    room_number VARCHAR(32),
    facility_name VARCHAR(32),
    address VARCHAR(500)
)
LANGUAGE plpgsql
AS $$
BEGIN
if not exists (SELECT
        c.room_number,
        f.facility_name,
        f.address
    FROM
        courses.classroom c
        JOIN courses.facility f ON c.facility_id = f.facility_id
        LEFT JOIN courses.timetable t ON c.classroom_id = t.classroom_id AND t.week_day = week_day_param
    WHERE
        t.classroom_id IS NULL AND c.type = 'лекционная аудитория') then
        RAISE NOTICE 'На день недели % нет свободных лекционных аудиторий', week_day_param;
    end if;

    RETURN QUERY
    SELECT
        c.room_number,
        f.facility_name,
        f.address
    FROM
        courses.classroom c
        JOIN courses.facility f ON c.facility_id = f.facility_id
        LEFT JOIN courses.timetable t ON c.classroom_id = t.classroom_id AND t.week_day = week_day_param
    WHERE
        t.classroom_id IS NULL AND c.type = 'лекционная аудитория';

END;
$$;
```

```
CREATE OR REPLACE FUNCTION GetFreeClassrooms(
    IN week_day_param VARCHAR(12)
)
RETURNS TABLE (
    room_number VARCHAR(32),
```

```
    facility_name VARCHAR(32),
    address VARCHAR(500)
)
LANGUAGE plpgsql
AS $$
BEGIN
if not exists (SELECT
        c.room_number,
        f.facility_name,
        f.address
    FROM
        courses.classroom c
        JOIN courses.facility f ON c.facility_id = f.facility_id
        LEFT JOIN courses.timetable t ON c.classroom_id = t.classroom_id AND t.week_day = week_day_param
    WHERE
        t.classroom_id IS NULL AND c.type = 'лекционная аудитория') then
        RAISE NOTICE 'На день недели % нет свободных лекционных аудиторий', week_day_param;
    end if;

    RETURN QUERY
    SELECT
        c.room_number,
        f.facility_name,
        f.address
    FROM
        courses.classroom c
        JOIN courses.facility f ON c.facility_id = f.facility_id
        LEFT JOIN courses.timetable t ON c.classroom_id = t.classroom_id AND t.week_day = week_day_param
    WHERE
        t.classroom_id IS NULL AND c.type = 'лекционная аудитория';

END;
$$;
```

```
COURSES=# SELECT * FROM GetFreeClassrooms('ПН');
 room_number |    facility_name    |                      address
-------------+---------------------+-----------------------------------------------------
 j16         | Корпус им. Голыгина | Россия, г. Санкт-Петербург, Юбилейная ул., д. 11
(1 row)

COURSES=# SELECT * FROM GetFreeClassrooms('ВТ');
NOTICE:  На день недели ВТ нет свободных лекционных аудиторий
 room_number | facility_name | address
-------------+---------------+---------
(0 rows)
```

4) Триггер для логирования событий вставки, удаления, редактирования данных в таблице ***Education***:

Создаем таблицу для логов:

```
create table if not exists courses.education_log
(
   log_id serial primary key,
   operation character varying(10),
   listener_number bigint,
   status character(32),
   passport bigint,
   group_name character varying,
   education_start date,
   education_end date,
   timestamp timestamp
);
```

Создаем функцию для логгирования и тригер на изменение таблицы ***Education***:

```
CREATE OR REPLACE FUNCTION log_education_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO courses.education_log (operation, listener_number, status, passport, group_name, education_start, education_end, timestamp)
        VALUES ('INSERT', NEW.listener_number, NEW.status, NEW.passport, NEW.group_name, NEW.education_start, NEW.education_end, current_timestamp);

    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO courses.education_log (operation, listener_number, status, passport, group_name, education_start, education_end, timestamp)
        VALUES ('UPDATE', NEW.listener_number, NEW.status, NEW.passport, NEW.group_name, NEW.education_start, NEW.education_end, current_timestamp);

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO courses.education_log (operation, listener_number, status, passport, group_name, education_start, education_end, timestamp)
        VALUES ('DELETE', OLD.listener_number, OLD.status, OLD.passport, OLD.group_name, OLD.education_start, OLD.education_end, current_timestamp);
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

```
create or replace function log_education_changes()
returns trigger as $$
begin
   if tg_op = 'insert' then
      insert into courses.education_log (operation, listener_number, status, passport, group_name, education_start,
education_end, timestamp)
      values ('insert', new.listener_number, new.status, new.passport, new.group_name, new.education_start,
new.education_end, current_timestamp);

   elsif tg_op = 'update' then
      insert into courses.education_log (operation, listener_number, status, passport, group_name, education_start,
education_end, timestamp)
      values ('update', new.listener_number, new.status, new.passport, new.group_name, new.education_start,
new.education_end, current_timestamp);

   elsif tg_op = 'delete' then
      insert into courses.education_log (operation, listener_number, status, passport, group_name, education_start,
education_end, timestamp)
      values ('delete', old.listener_number, old.status, old.passport, old.group_name, old.education_start,
old.education_end, current_timestamp);
   end if;

   return null;
end;
$$ language plpgsql;

create trigger education_changes_trigger
after insert or update or delete
on courses.education
for each row
execute function log_education_changes();
```

Удаляем одну строку, потом я добавила сроки обучения в таблицу, и вот результаты логирования (содержание таблицы education_log):

```
COURSES=# SELECT * FROM courses.education_log;
 log_id | operation | listener_number |              status              |  passport  | group_name | education_start | education_end |         timestamp
--------+-----------+-----------------+----------------------------------+------------+------------+-----------------+---------------+---------------------------
      1 | DELETE    |          888810 | обучается                        | 2222222277 | K33        |                 |               | 2024-01-17 20:23:42.153981
      2 | UPDATE    |          888881 | обучается                        | 2222222222 | M11        | 2021-02-02      |               | 2024-01-17 20:26:37.434891
      3 | UPDATE    |          888881 | обучается                        | 2222222222 | M11        | 2021-02-02      | 2023-02-02    | 2024-01-17 20:27:54.347453
      4 | UPDATE    |          888882 | обучается                        | 2222222233 | M11        | 2021-02-02      | 2023-02-02    | 2024-01-17 20:27:54.347453
      5 | UPDATE    |          888883 | обучается                        | 2231555399 | M11        | 2021-02-02      | 2023-02-02    | 2024-01-17 20:27:54.347453
```

...

Вывод:

В ходе выполнения лабораторной работы были разработаны и выполнены запросы на выборку данных, а также были созданы представления для базы данных PostgreSQL в соответствии с поставленной индивидуальной задачей. Кроме того, мы успешно реализовали разнообразные запросы на модификацию данных. Провели создание как простых, так и составных индексов, а также проанализировали время выполнения запросов при их использовании.