МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

Лабораторная работа №2
по курсу Постреляционные базы данных

Тема: «Постреляционное расширение языка SQL на примере PostgreSQL»

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-22М _____
Сметанкин К.И.
"__"_____2020 г.

ПРЕПОДАВАТЕЛЬ:

Виноградова М.В. _____
к.т.н., доцент
"__"_____2020 г.

Москва – 2020

Задание 1 *Базовая часть (удовлетворительно)*

1.1 *Создание и заполнение таблицы*

```sql
CREATE EXTENSION IF NOT EXISTS CITEXT;

DROP TABLE IF EXISTS users, posts, tag, comments;

CREATE TYPE fullname AS
(
    firstname  CITEXT,
    lastname   CITEXT,
    middlename CITEXT
);

CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');

CREATE TABLE IF NOT EXISTS users
(
    id         SERIAL PRIMARY KEY,
    login      CITEXT   NOT NULL UNIQUE,
    password   TEXT,
    name       fullname NOT NULL DEFAULT ('empty', 'empty', 'empty')::fullname,
    avatar     CITEXT   NOT NULL,
    karma      INT               DEFAULT 0,
    registered timestamptz       DEFAULT now(),
    mood       mood              DEFAULT 'ok'
);

ALTER TABLE users
    ADD CHECK ( length(password) > 5 );

CREATE TYPE super_users_level AS ENUM ('base', 'ok', 'incredible');

CREATE TABLE IF NOT EXISTS super_users
(
    level super_users_level DEFAULT 'base'
)
    INHERITS (users);

CREATE TABLE IF NOT EXISTS posts
(
    id          SERIAL PRIMARY KEY,
    header      CITEXT   NOT NULL UNIQUE,
    short_topic CITEXT   NOT NULL UNIQUE,
    main_topic  CITEXT   NOT NULL UNIQUE,
    user_id     int      NOT NULL REFERENCES users (id),
    authors     CITEXT[] NOT NULL,
    show        bool        DEFAULT False,
    created     timestamptz DEFAULT now()
);

CREATE TABLE IF NOT EXISTS tag
(
    id   SERIAL PRIMARY KEY,
    name CITEXT NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS comments
(
    id        SERIAL PRIMARY KEY,
    parent_id int REFERENCES comments (id),
    user_id   int  NOT NULL REFERENCES users (id),
    post_id   int  NOT NULL REFERENCES posts (id),
    payload   text NOT NULL,
    show      bool        DEFAULT True,
    created   timestamptz DEFAULT now()
);
```

1.2 *Программирование функций с применением SQL\PSM*

1. Создать скалярную функцию

Получаем количество пользователей с заданным настроением

```
-- получает количество пользователей с заданным настроением
CREATE OR REPLACE FUNCTION users_with_mood (
    mood
)
    RETURNS SETOF BIGINT
    AS $$
BEGIN
    RETURN QUERY
    SELECT
        count(id)
    FROM
        users
    WHERE
        mood = $1;
END
$$
LANGUAGE plpgsql;
```

Использование

```
SELECT
    *
FROM
    users_with_mood ('ok');
```

2. Создать табличную функцию

возвращает таблицу с инфой о созданных постах

```
CREATE OR REPLACE FUNCTION get_topics_short_info (
    user_id integer
)
    RETURNS TABLE (
        id integer,
        header CITEXT,
        short_topic CITEXT,
        SHOW BOOLEAN,
        created timestamptz
    )
    AS $$
BEGIN
    RETURN QUERY
    SELECT
        p.id,
        p.header,
        p.short_topic,
        p.show,
        p.created
    FROM
        posts p
        JOIN users u ON p.user_id = u.id
    WHERE
        u.id = $1;
END
$$
LANGUAGE plpgsql;
```

пример

```
SELECT * FROM get_topics_short_info(1);
```

id (integer)	header (citext)	short_topic (citext)	show (boolean)	created (timestamp with time zone)
1	6	test_header_2	short_2	false
2	1	test_header	short	false

3. Создать хранимую процедуру

Создает пост с авторством пользователя

```
CREATE OR REPLACE PROCEDURE create_post (
    header citext,
    short_topic citext,
    main_topic citext,
    authors citext[],
    username citext
)
LANGUAGE plpgsql
AS $$
DECLARE
    userID int;
BEGIN
    SELECT
        id INTO STRICT userID
    FROM
        users
    WHERE
        LOGIN = create_post.username;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE EXCEPTION 'User % not found', create_post.username

    INSERT INTO posts (header, short_topic, main_topic, user_id, authors)
        VALUES (create_post.header, create_post.short_topic, create_post.main_topic, us
    COMMIT;
END;
$$;
```

пример

```
CALL create_post ('kek', 'kek1', 'kek2', '{"e", "w", "d"}', 'smet_k');
```

Задание 2 *Расширенная часть (хорошо)*

2.1 Извлечение части записей и результатов запросов на изменение данных

Продемонстрировать выполнение запроса на получение первых 3-х записей из результата (limit).

```
SELECT
    *
FROM
    posts
LIMIT 3;
```

Продемонстрировать выполнение запроса на добавление/изменение данных с отображением измененных строк (returning).

```
UPDATE
    posts p
SET
    header = 'test_header_1_updated'
WHERE
    id = 5
RETURNING
    p.*;
```

2.2 Выполнение рекурсивных запросов

Продемонстрировать выполнение рекурсивного запроса

```
WITH RECURSIVE r AS (
    SELECT
        1 AS i,
        1 AS factorial
    UNION
    SELECT
        i + 1 AS i,
        factorial * (i + 1) AS factorial
    FROM
        r
)
SELECT
    factorial
FROM
    r
LIMIT 12;
```

2.3 Создание динамических запросов

Создать хранимую процедуру с динамическим запросом

Проверяет присутствие в авторах, если отстутствует, то добавляет пользователя

```
CREATE OR REPLACE PROCEDURE create_valid_post (
    header citext,
    short_topic citext,
    main_topic citext,
    authors citext[],
    username citext
)
LANGUAGE plpgsql
AS $$
DECLARE
    authors citext[];
    userID int;
    author citext;
BEGIN
    SELECT
        id INTO STRICT userID
    FROM
        users
    WHERE
        LOGIN = create_valid_post.username;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'User % not found', create_valid_post.username;
    END IF;
    authors = create_valid_post.authors;
    author = '';
    IF NOT create_valid_post.username = ANY (authors) THEN
        authors = array_append(authors, create_valid_post.username);
    END IF;
    INSERT INTO posts (header, short_topic, main_topic, user_id, authors)
        VALUES (create_valid_post.header, create_valid_post.short_topic, create_valid_p
    COMMIT;
END;
$$;
```