**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления КАФЕДРА Системы обработки информации и управления

**Лабораторная работа №2**

**«»**

ИСПОЛНИТЕЛИ: Гаврилов Л.Я.

ФИО

группа ИУ5-24М

подпись

"" март 2024 г.

ПРЕПОДАВАТЕЛЬ: Виноградова М.В.

ФИО

подпись

" " 2024 г.

Москва – 2024

1. **Цель работы**

* Изучить постреляционные возможности языка SQL [1].
* Освоить языки и технологии SQL\PSM на примере PostgreSQL [2].
* Получить навыки программирования на стороне сервера

1. **Задание**
2. В среде PgAdmin (PostgreSQL) создать БД. В БД создать две-три таблицы по теме, "Подбор фильмов на основе глубокого обучения", открыть таблицы на редактирование и заполнить тестовыми данными.
3. Создать скалярную функцию. Вызвать функцию из окна запроса.
4. Создать табличную функцию (inline). Вызвать функцию из окна запроса.
5. Создать табличную функцию (multi-statement). Продемонстрировать наполнение результирующего множества записей. Вызвать функцию из окна запроса.
6. Создать хранимую процедуру, содержащую запросы, вызов и перехват исключений. Вызвать процедуру из окна запроса. Проверить перехват и создание исключений.
7. Продемонстрировать в функциях и процедурах работу условных операторов и выполнение динамического запроса

# Порядок выполнения

1 В среде PgAdmin (PostgreSQL) создать БД. В БД создать две-три таблицы по теме, "Подбор фильмов на основе глубокого обучения", связанные как один-ко-многим, содержащие первичные и внешние ключи. Открыть таблицы на редактирование и заполнить тестовыми данными.

**SQL**

CREATE DATABASE MovieDB;

\c MovieDB;

CREATE TABLE Movies (

MovieID serial PRIMARY KEY,

Title VARCHAR (100),

Genre VARCHAR (100),

Rating FLOAT

);

CREATE TABLE Users (

UserID serial PRIMARY KEY,

Name VARCHAR (100),

Age INT

);

CREATE TABLE Ratings (

UserID INT REFERENCES Users(UserID),

MovieID INT REFERENCES Movies(MovieID),

Rating INT

);

2 Создать скалярную функцию. Вызвать функцию из окна запроса.

**SQL**

CREATE FUNCTION get\_movie\_count() RETURNS INT AS $$

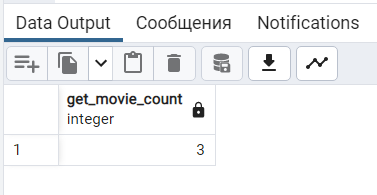
BEGIN

RETURN (SELECT COUNT(\*) FROM Movies);

END;

$$ LANGUAGE plpgsql;

SELECT get\_movie\_count();



3 Создать табличную функцию (inline). Вызвать функцию из окна запроса.

**SQL**

CREATE FUNCTION get\_movies() RETURNS TABLE (MovieID INT, Title character varying(100)) AS $$

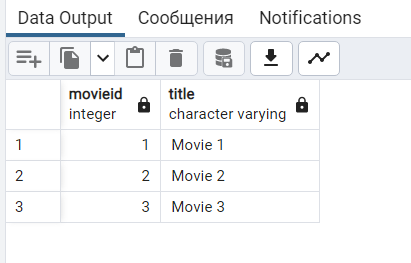
BEGIN

RETURN QUERY SELECT Movies.MovieID, Movies.Title FROM Movies;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM get\_movies();



4 Создать табличную функцию (multi-statement). Продемонстрировать

наполнение результирующего множества записей. Вызвать функцию из окна

запроса.

**SQL**

CREATE FUNCTION get\_movies\_with\_ratings() RETURNS TABLE (MovieID INT, Title character varying(100), AvgRating FLOAT) AS $$

DECLARE

temp\_table RECORD;

BEGIN

CREATE TEMP TABLE temp\_ratings AS SELECT Movies.MovieID, Movies.Title, AVG(Ratings.Rating) AS AvgRating FROM Movies JOIN Ratings ON Movies.MovieID = Ratings.MovieID GROUP BY Movies.MovieID, Movies.Title;

FOR temp\_table IN SELECT \* FROM temp\_ratings

LOOP

MovieID := temp\_table.MovieID;

Title := temp\_table.Title;

AvgRating := temp\_table.AvgRating;

RETURN NEXT;

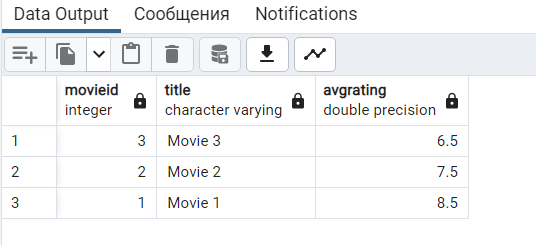
END LOOP;

DROP TABLE temp\_ratings;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM get\_movies\_with\_ratings();



5 Создать хранимую процедуру, содержащую запросы, вызов и перехват

исключений. Вызвать процедуру из окна запроса. Проверить перехват и

создание исключений.

**SQL**

CREATE PROCEDURE insert\_movie(IN title TEXT, IN genre TEXT, IN rating FLOAT) LANGUAGE plpgsql AS $$

BEGIN

IF rating < 0 OR rating > 10 THEN

RAISE EXCEPTION 'Rating must be between 0 and 10';

ELSE

INSERT INTO Movies (Title, Genre, Rating) VALUES (title, genre, rating);

END IF;

END;

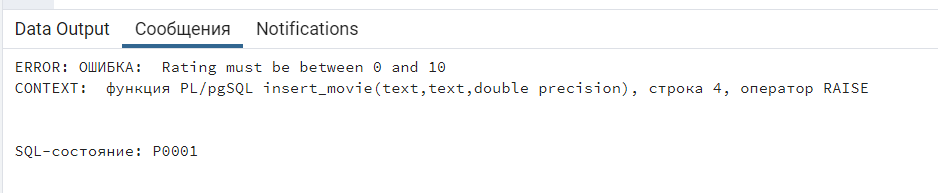
$$;

CALL insert\_movie('New Movie', 'Drama', 8.5);

Чтобы проверить перехват исключений, вы можете вызвать процедуру с недопустимым рейтингом, например:

**SQL**

CALL insert\_movie('New Movie', 'Drama', 11);



Этот вызов должен вызвать исключение с сообщением “Rating must be between 0 and 10”, потому что рейтинг 11 выходит за допустимый диапазон.

6 Продемонстрировать в функциях и процедурах работу условных операторов

и выполнение динамического запроса

**SQL**

DROP FUNCTION IF EXISTS dynamic\_query(text);

CREATE OR REPLACE FUNCTION dynamic\_query(p\_tablename text) RETURNS SETOF RECORD AS $$

DECLARE

query text;

BEGIN

IF p\_tablename = 'Movies' THEN

RETURN QUERY EXECUTE 'SELECT \* FROM Movies';

ELSIF p\_tablename = 'Users' THEN

RETURN QUERY EXECUTE 'SELECT \* FROM Users';

ELSE

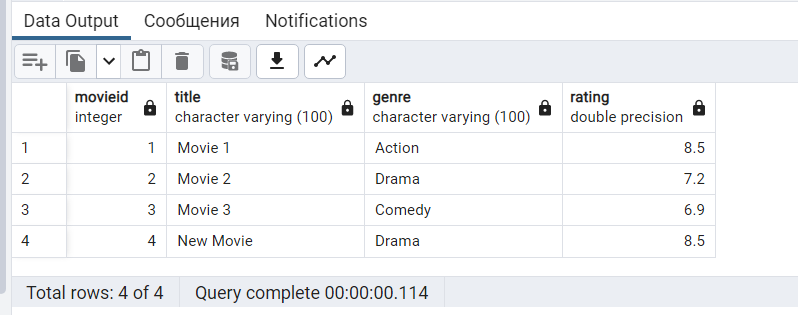
RAISE EXCEPTION 'Invalid table name';

END IF;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM dynamic\_query('Movies') AS t(MovieID INT, Title character varying(100), Genre character varying(100), Rating FLOAT);



функция, которая выбирает все фильмы с рейтингом в указанном диапазоне:

CREATE OR REPLACE FUNCTION dynamic\_query(p\_tablename text, p\_columnname text, p\_lowerbound float, p\_upperbound float) RETURNS SETOF RECORD AS $$

DECLARE

query text;

BEGIN

query := format('SELECT \* FROM %I WHERE %I BETWEEN %L AND %L', p\_tablename, p\_columnname, p\_lowerbound, p\_upperbound);

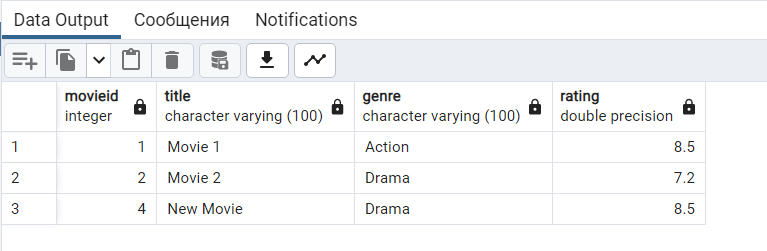
RETURN QUERY EXECUTE query;

END;

$$ LANGUAGE plpgsql;

Вызов в следующем виде  
SELECT \* FROM dynamic\_query('Movies', 'Rating', 7.0, 9.0) AS t(MovieID INT, Title character varying(100), Genre character varying(100), Rating FLOAT);

Этот код выберет все фильмы с рейтингом между 7.0 и 9.0



Скалярная

CREATE OR REPLACE FUNCTION get\_average\_rating(p\_movieid INT) RETURNS FLOAT AS $$

DECLARE

avg\_rating FLOAT;

BEGIN

SELECT AVG(Rating) INTO avg\_rating FROM Ratings WHERE MovieID = p\_movieid;

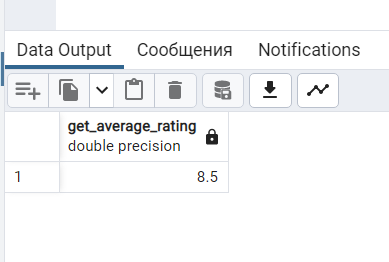
RETURN avg\_rating;

END;

$$ LANGUAGE plpgsql;

SELECT get\_average\_rating(1);

средний рейтинг для фильма с определенным ID.



1. Выбор всех фильмов:

**SQL**

SELECT \* FROM dynamic\_query('Movies') AS t(MovieID INT, Title character varying(100), Genre character varying(100), Rating FLOAT);

1. Выбор всех пользователей:

**SQL**

SELECT \* FROM dynamic\_query('Users') AS t(UserID INT, Name character varying(100), Age INT);

Динамичческий запрос в ручную вбивать нижний и верхний порог