**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 №1**

**«»**

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 объектно-реляционную модель данных [1] и возможности языка SQL для работы с ней;
* Освоить построение запросов на языке SQL для описания ограничений целостности, определения и использования полей сложных типов данных и наследования таблиц;
* Получить навыки создания объектно-реляционной базы данных и выполнения запросов к ней на примере СУБД PostgreSQL [2] в среде PgAdmin.

1. **Задание**
2. 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 и ограничения: уникальности, Not NULL, на значения, на значение по умолчанию, автоинкремент. Открыть таблицы на редактирование и заполнить тестовыми данными. Проверить действие ограничений.
3. Добавить в таблицу поле типа массив и продемонстрировать работу с ним.
4. Добавить в таблицу поле типа структуры на основе структурного типа и продемонстрировать работу с ним.
5. Создать собственный перечислимый тип и добавить, как поле в таблицу. Продемонстрировать работу с типом.
6. Создать производную таблицу с дополнительным полем. Продемонстрировать работу с базовыми и производными таблицами (CRUD). Вывести отдельно записи базовой и производной таблиц, удовлетворяющих некоторому условию.

# Порядок выполнения

*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 и ограничения: уникальности, Not NULL, на значения, на значение по умолчанию, автоинкремент. Открыть таблицы на редактирование и заполнить тестовыми данными. Проверить действие ограничений.*

1. **Создание БД в PgAdmin**

**SQL**

CREATE DATABASE movies;

1. **Создание таблиц**

**SQL**

USE movies;

CREATE TABLE Genres (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL UNIQUE

);

CREATE TABLE Movies (

id SERIAL PRIMARY KEY,

title VARCHAR(100) NOT NULL,

genre\_id INT,

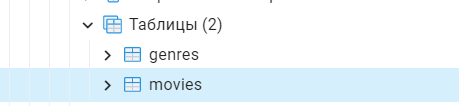
release\_year INT DEFAULT 2024,

CONSTRAINT fk\_genre

FOREIGN KEY(genre\_id)

REFERENCES Genres(id)

);



1. **Заполнение таблиц тестовыми данными**

**SQL**

INSERT INTO Genres (name) VALUES ('Комедия'), ('Драма'), ('Фантастика');

INSERT INTO Movies (title, genre\_id) VALUES ('Фильм 1', 1), ('Фильм 2', 2), ('Фильм 3', 3);

1. **Проверка действия ограничений**

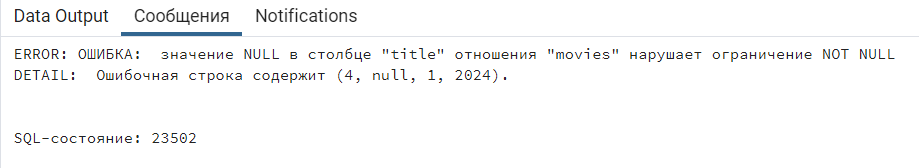
**SQL**

-- Попытка вставить дубликат в поле 'name' таблицы 'Genres' вызовет ошибку, потому что мы установили ограничение UNIQUE

INSERT INTO Genres (name) VALUES ('Комедия');

-- Попытка вставить NULL в поле 'title' таблицы 'Movies' вызовет ошибку, потому что мы установили ограничение NOT NULL

INSERT INTO Movies (title, genre\_id) VALUES (NULL, 1);



Во втором пункте мы создали две таблицы: Genres и Movies. Они связаны между собой отношением “один ко многим”. Одному жанру (Genres) может соответствовать множество фильмов (Movies). Это отношение реализовано через внешний ключ genre\_id в таблице Movies, который ссылается на поле id в таблице Genres.

Теперь давай разберем код:

1. CREATE DATABASE movies; - создает новую базу данных с именем movies.
2. USE movies; - переключает текущую базу данных на movies.
3. CREATE TABLE Genres... - создает таблицу Genres с двумя полями: id и name. id - это первичный ключ, который автоматически увеличивается при каждой новой записи. name - это уникальное название жанра.
4. CREATE TABLE Movies... - создает таблицу Movies с четырьмя полями: id, title, genre\_id и release\_year. id - это первичный ключ, который автоматически увеличивается при каждой новой записи. title - это название фильма, которое не может быть NULL. genre\_id - это внешний ключ, который ссылается на поле id в таблице Genres. release\_year - это год выпуска фильма, по умолчанию установлен в 2024.
5. INSERT INTO Genres... - вставляет тестовые данные в таблицу Genres.
6. INSERT INTO Movies... - вставляет тестовые данные в таблицу Movies.
7. Последние два запроса INSERT INTO... - это попытки нарушить ограничения, которые мы установили для таблиц. Они вызовут ошибки, что позволит проверить, работают ли ограничения правильно.

*Добавить в таблицу поле типа массив и продемонстрировать работу с ним.*

1. **Добавление поля в таблицу**

**SQL**

ALTER TABLE Movies ADD COLUMN actors TEXT[];

1. **Вставка данных в новое поле**

**SQL**

UPDATE Movies SET actors = ARRAY['Актер 1', 'Актер 2', 'Актер 3'] WHERE id = 1;

UPDATE Movies SET actors = ARRAY['Актер 4', 'Актер 5', 'Актер 6'] WHERE id = 2;

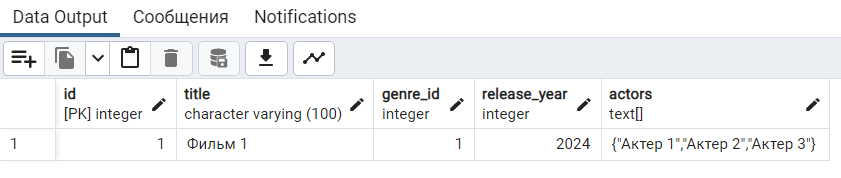
UPDATE Movies SET actors = ARRAY['Актер 7', 'Актер 8', 'Актер 9'] WHERE id = 3;

1. **Проверка работы с массивом**

**SQL**

-- Выборка фильмов, в которых снимался 'Актер 1'

SELECT \* FROM Movies WHERE 'Актер 1' = ANY(actors);



*Добавить в таблицу поле типа структуры на основе структурного типа и продемонстрировать работу с ним.*

1. **Создание пользовательского типа данных**

**SQL**

CREATE TYPE actor AS (

name VARCHAR(100),

age INT

);

1. **Добавление нового поля в таблицу**

**SQL**

ALTER TABLE Movies ADD COLUMN lead\_actor actor;

1. **Вставка данных в новое поле**

**SQL**

UPDATE Movies SET lead\_actor = ROW('Актер 1', 30) WHERE id = 1;

UPDATE Movies SET lead\_actor = ROW('Актер 2', 40) WHERE id = 2;

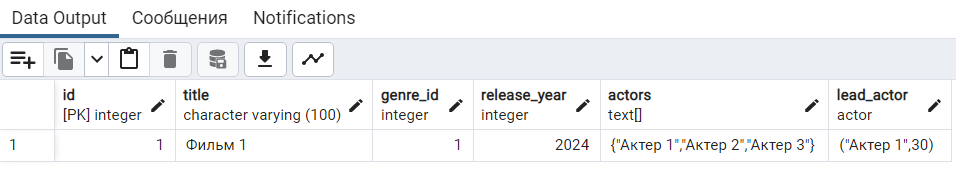
UPDATE Movies SET lead\_actor = ROW('Актер 3', 50) WHERE id = 3;

1. **Проверка работы с пользовательским типом данных**

**SQL**

-- Выборка фильмов, в которых главную роль играл 'Актер 1'

SELECT \* FROM Movies WHERE (lead\_actor).name = 'Актер 1';



*Создать собственный перечислимый тип и добавить, как поле в таблицу. Продемонстрировать работу с типом.*

1. **Создание перечислимого типа**

**SQL**

CREATE TYPE movie\_rating AS ENUM ('Очень плохо', 'Плохо', 'Средне', 'Хорошо', 'Отлично');

1. **Добавление нового поля в таблицу**

**SQL**

ALTER TABLE Movies ADD COLUMN rating movie\_rating;

1. **Вставка данных в новое поле**

**SQL**

UPDATE Movies SET rating = 'Хорошо' WHERE id = 1;

UPDATE Movies SET rating = 'Отлично' WHERE id = 2;

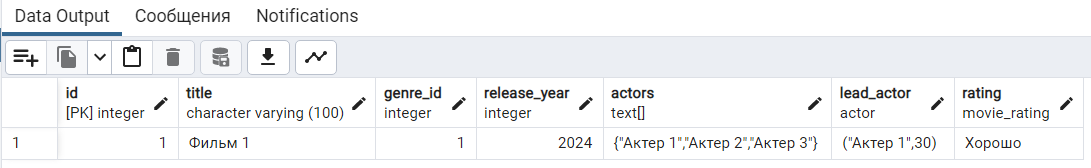
UPDATE Movies SET rating = 'Средне' WHERE id = 3;

1. **Проверка работы с перечислимым типом**

**SQL**

-- Выборка фильмов с рейтингом 'Хорошо'

SELECT \* FROM Movies WHERE rating = 'Хорошо';



*Создать производную таблицу с дополнительным полем. Продемонстрировать работу с базовыми и производными таблицами (CRUD). Вывести отдельно записи базовой и производной таблиц, удовлетворяющих некоторому условию.*

Создадим производную таблицу MoviesExtended на основе таблицы Movies и добавим в нее дополнительное поле budget.

1. **Создание производной таблицы**

**SQL**

CREATE TABLE MoviesExtended (

id INT,

budget INT,

PRIMARY KEY (id),

FOREIGN KEY (id) REFERENCES Movies (id)

);

1. **Вставка данных в новую таблицу**

**SQL**

INSERT INTO MoviesExtended (id, budget) VALUES (1, 1000000), (2, 2000000), (3, 3000000);

1. **Проверка работы с базовой и производной таблицей**

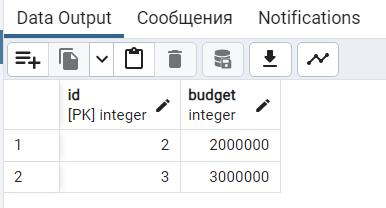
**SQL**

-- Выборка записей из базовой таблицы, удовлетворяющих условию

SELECT \* FROM Movies WHERE rating = 'Хорошо';

-- Выборка записей из производной таблицы, удовлетворяющих условию

SELECT \* FROM MoviesExtended WHERE budget > 1500000;



1. **Обновление данных в таблицах**

**SQL**

-- Обновление записи в базовой таблице

UPDATE Movies SET title = 'Новый фильм' WHERE id = 1;

-- Обновление записи в производной таблице

UPDATE MoviesExtended SET budget = 2500000 WHERE id = 1;

1. **Удаление данных из таблиц**

**SQL**

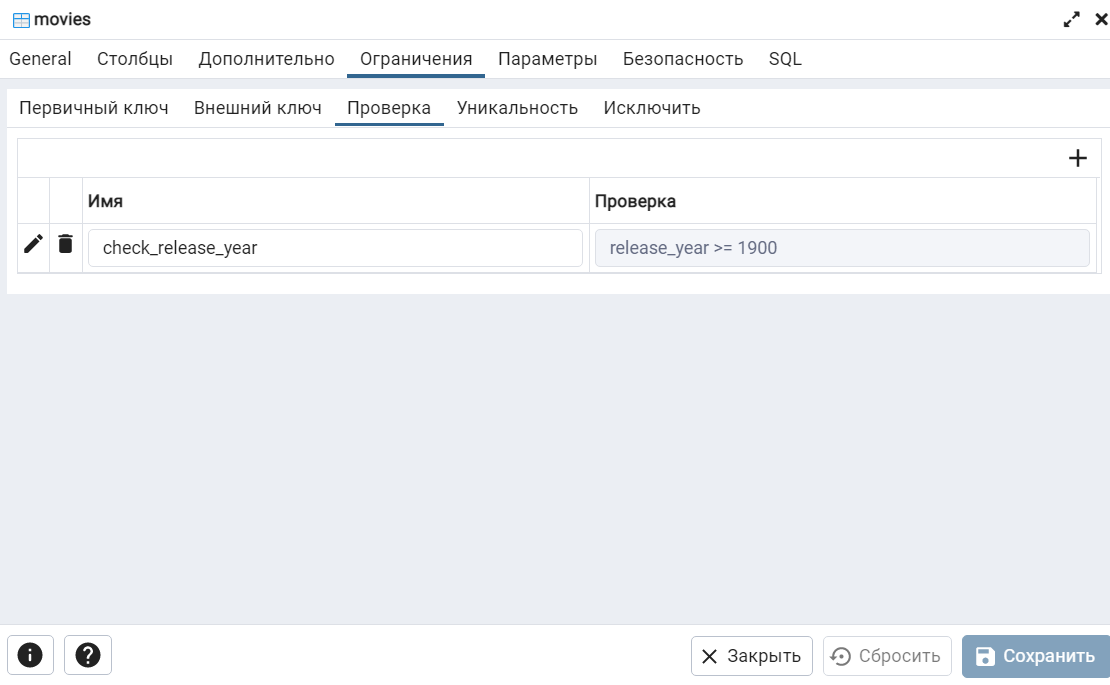
-- Удаление записи из производной таблицы

DELETE FROM MoviesExtended WHERE id = 1;

-- Удаление записи из базовой таблицы

DELETE FROM Movies WHERE id = 1;

1 Через мастер добавить ограничения общего вида chek



2 Через Alter Table сделать чтобы внешний ключ с каскадным обновлением и удалением

-- Удаление существующего внешнего ключа

ALTER TABLE Movies DROP CONSTRAINT fk\_genre;

-- Добавление нового внешнего ключа с каскадными операциями

ALTER TABLE Movies ADD CONSTRAINT fk\_genre FOREIGN KEY (genre\_id) REFERENCES Genres(id) ON UPDATE CASCADE ON DELETE CASCADE;

1. **Создание (INSERT): Добавление новой записи в таблицу.**

**SQL**

INSERT INTO Movies (title, genre\_id, release\_year, rating)

VALUES ('Новый фильм', 1, 2024, 'Хорошо');

1. **Чтение (SELECT): Получение данных из таблицы.**

**SQL**

SELECT \* FROM Movies;

1. **Обновление (UPDATE): Изменение существующих данных в таблице.**

**SQL**

UPDATE Movies

SET title = 'Обновленный фильм', rating = 'Отлично'

WHERE id = 1;

1. **Удаление (DELETE): Удаление существующих данных из таблицы.**

**SQL**

DELETE FROM Movies WHERE id = 1;

1. **Геометрические типы: PostgreSQL, такие как точки, линии, окружности и многоугольники. Например, добавить поле location типа POINT в таблицу Movies:**

**SQL**

ALTER TABLE Movies ADD COLUMN location POINT;

Вставить данные в это поле:

**SQL**

UPDATE Movies SET location = POINT(40.7128, 74.0060) WHERE id = 1;

1. **JSON: PostgreSQL поддерживает типы данных JSON и JSONB. Вы можете добавить поле details типа JSON в таблицу Movies:**

**SQL**

ALTER TABLE Movies ADD COLUMN details JSON;

Вставить данные в это поле:

**SQL**

UPDATE Movies SET details = '{"director": "Режиссер 1", "duration": 120}' WHERE id = 1;

1. **Диапазоны: PostgreSQL поддерживает диапазоны числовых, временных и других типов. Вы можете добавить поле showtimes типа TSRANGE (диапазон временных меток) в таблицу Movies:**

**SQL**

ALTER TABLE Movies ADD COLUMN showtimes TSRANGE;

Вставить данные в это поле:

**SQL**

UPDATE Movies SET showtimes = TSRANGE('2024-01-01 14:30', '2024-01-01 16:30') WHERE id = 1;

Расстояние между моей точкой и точкой 1.1

евклидово расстояние между двумя точками (40.7128, 74.0060) и (1, 1) в PostgreSQL, вы можете использовать следующий SQL-запрос:

**SQL**

SELECT point '(40.7128, 74.0060)' <-> point '(1, 1)';