МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ
МЕГАФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Создание информационной системы для каталогизации цифровой фильмотеки «MyFilmList»

курсовая работа, этап №2

по дисциплине «Информационные системы»
направления бакалавриата 09.03.04 «Программная инженерия»,
образовательное направление «Системное и прикладное программное обеспечение»

Научный руководитель:

Егошин Алексей Васильевич

Работу выполнили:

Студенты 3 курса

Группы P3315

Барсуков Максим Андреевич

Горляков Даниил Петрович

г. Санкт-Петербург – 2024

Оглавление

Задание

1. Сформировать ER-модель базы данных (на основе описаний предметной области и прецедентов из предыдущего этапа).
   ER-модель должна:
   a. включать в себя не менее 10 сущностей;
   b. содержать хотя бы одно отношение вида «многие-ко-многим».
2. Согласовать ER-модель с преподавателем. На основе ER-модели построить даталогическую модель.
3. Реализовать даталогическую модель в реляционной СУБД PostgreSQL.
4. Обеспечить целостность данных при помощи средств языка DDL и триггеров.
5. Реализовать скрипты для создания, удаления базы данных, заполнения базы тестовыми данными.
6. Предложить pl/pgsql-функции и процедуры, для выполнения критически важных запросов (которые потребуются при последующей реализации прецедентов).
7. Создать индексы на основе анализа использования базы данных в контексте описанных на первом этапе прецедентов. Обосновать полезность созданных индексов для реализации представленных на первом этапе бизнес-процессов.
8. Составить отчет.

1. ER-модель

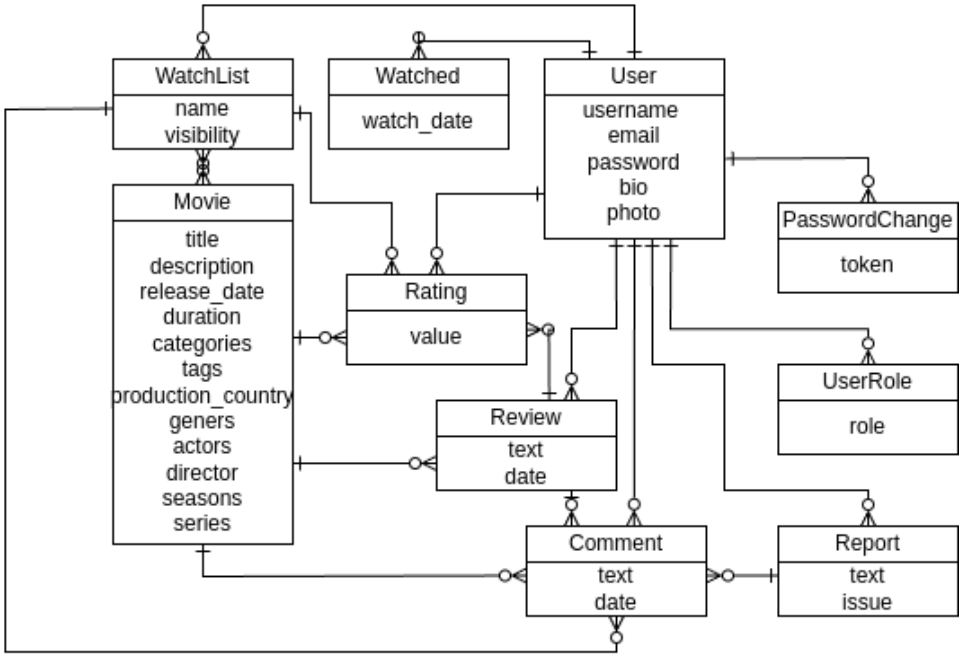

Рисунок 1 — ER-модель

2. Даталогическая модель

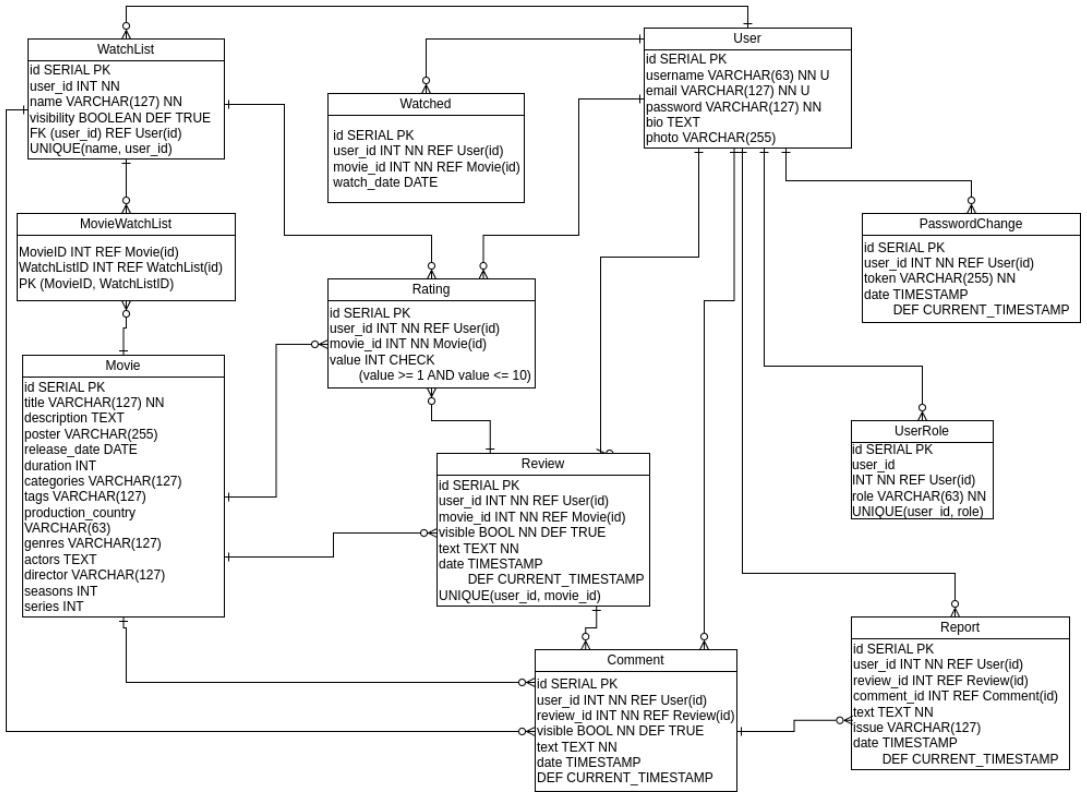

Рисунок 2 — Даталогическая модель

3. Реализация даталогической модели в реляционной СУБД PostgreSQL

```
CREATE TABLE "User" (
    id SERIAL PRIMARY KEY,
    username VARCHAR(63) NOT NULL UNIQUE,
    email VARCHAR(127) NOT NULL UNIQUE,
    password VARCHAR(127) NOT NULL,
    bio TEXT,
    photo VARCHAR(255)
);

CREATE TABLE "WatchList" (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES "User"(id) ON DELETE SET NULL,
    name VARCHAR(127) NOT NULL,
    visibility BOOLEAN NOT NULL DEFAULT TRUE,
    UNIQUE(name, user_id)
);

CREATE TABLE "Movie" (
    id SERIAL PRIMARY KEY,
    title VARCHAR(127) NOT NULL,
    description TEXT,
    poster VARCHAR(255)
    release_date DATE,
    duration INT,
    rating FLOAT,
    categories VARCHAR(127),
    tags VARCHAR(127),
    production_country VARCHAR(63),
    genres VARCHAR(127),
    actors TEXT,
    director VARCHAR(127),
    seasons INT CHECK (value IS NULL OR value >= 0),
    series INT CHECK (value IS NULL OR value >= 0)
);

CREATE TABLE "MovieWatchList" (
    MovieID INT NOT NULL REFERENCES "Movie"(id) ON DELETE CASCADE,
    WatchListID INT NOT NULL REFERENCES "WatchList"(id) ON DELETE CASCADE,
    PRIMARY KEY (MovieID, WatchListID)
);

CREATE TABLE "Watched" (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES "User"(id) ON DELETE CASCADE,
    movie_id INT NOT NULL REFERENCES "Movie"(id) ON DELETE CASCADE,
    watch_date DATE CHECK (value IS NULL OR value > CURRENT_DATE)
);
```

```
CREATE TABLE "Rating" (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES "User"(id) ON DELETE SET NULL,
    movie_id INT NOT NULL REFERENCES "Movie"(id) ON DELETE CASCADE,
    value INT CHECK (value >= 1 AND value <= 10)
);

CREATE TABLE "Review" (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES "User"(id) ON DELETE SET NULL,
    movie_id INT NOT NULL REFERENCES "Movie"(id) ON DELETE CASCADE,
    visible BOOLEAN NOT NULL DEFAULT TRUE,
    text TEXT NOT NULL,
    date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, movie_id)
);

CREATE TABLE "Comment" (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES "User"(id) ON DELETE SET NULL,
    review_id INT NOT NULL REFERENCES "Review"(id) ON DELETE CASCADE,
    visible BOOLEAN NOT NULL DEFAULT TRUE,
    text TEXT NOT NULL,
    date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE "Report" (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES "User"(id) ON DELETE SET NULL,
    review_id INT REFERENCES "Review"(id) ON DELETE RESTRICT,
    comment_id INT REFERENCES "Comment"(id) ON DELETE RESTRICT,
    text TEXT NOT NULL,
    issue VARCHAR(127),
    date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE "PasswordChange" (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES "User"(id) ON DELETE CASCADE,
    token VARCHAR(255) NOT NULL,
    date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE "UserRole" (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES "User"(id) ON DELETE CASCADE,
    role VARCHAR(63) NOT NULL,
    UNIQUE(user_id, role)
);
```

4. Триггеры

```
-- delete password's token after 7 days
CREATE OR REPLACE FUNCTION cleanup_password_tokens()
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM "PasswordChange" WHERE date < CURRENT_TIMESTAMP - INTERVAL '7
days';
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER cleanup_tokens
AFTER INSERT OR UPDATE ON "PasswordChange"
FOR EACH ROW
EXECUTE FUNCTION cleanup_password_tokens();


-- update movie rating
CREATE OR REPLACE FUNCTION update_movie_rating()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE "Movie"
    SET rating = (
        SELECT AVG(value)::NUMERIC(3, 2)
        FROM "Rating"
        WHERE movie_id = NEW.movie_id
    )
    WHERE id = NEW.movie_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER recalculate_movie_rating
AFTER INSERT OR UPDATE OR DELETE ON "Rating"
FOR EACH ROW
EXECUTE FUNCTION update_movie_rating();
```

5. Скрипты для создания, удаления БД, заполнения базы тестовыми данными

Скрипт для создания БД:

```
CREATE DATABASE movie_db;

\c movie_db;

-- Далее см. “Реализация даталогической модели в реляционной СУБД PostgreSQL”
```

Скрипт для удаления БД:

```
-- Завершение активных соединений с базой
SELECT pg_terminate_backend(pg_stat_activity.pid)
FROM pg_stat_activity
WHERE pg_stat_activity.datname = 'movie_db'
  AND pid <> pg_backend_pid();

-- Проверка наличия активных сессий после завершения
SELECT * FROM pg_stat_activity WHERE datname = 'movie_db';

-- Удаление базы данных
DROP DATABASE movie_db;
```

Скрипт для заполнения базы тестовыми данными:

```
INSERT INTO "User" (username, email, password, bio, photo)
VALUES
    ('john_doe', 'john@example.com', 'password123', 'Just a movie lover',
NULL),
    ('jane_smith', 'jane@example.com', 'securepass', 'Critic and reviewer',
NULL),
    ('admin_user', 'admin@example.com', 'adminpass', 'Admin of the
platform',NULL);

INSERT INTO "UserRole" (user_id, role)
VALUES
    (1, 'USER'),
    (2, 'USER'),
    (3, 'ADMIN');

INSERT INTO "Movie" (title, description, release_date, duration, categories,
tags, production_country, genres, actors, director, seasons, series)
VALUES
    ('Inception', 'A mind-bending thriller', '2010-07-16', 148, 'Sci-Fi',
'dreams, mind', 'USA', 'Science Fiction', 'Leonardo DiCaprio, Ellen Page',
'Christopher Nolan', NULL, NULL),
```

```
    ('Breaking Bad', 'A high school teacher turns to cooking meth',
'2008-01-20', 60, 'Drama', 'crime, drugs', 'USA', 'Crime, Drama', 'Bryan
Cranston, Aaron Paul', 'Vince Gilligan', 5, 62);

INSERT INTO "WatchList" (user_id, name, visibility)
VALUES
    (1, 'Favorite Movies', TRUE),
    (2, 'To Watch', TRUE);

INSERT INTO "MovieWatchList" (MovieID, WatchListID)
VALUES
    (1, 1), -- Inception in "Favorite Movies"
    (2, 2); -- Breaking Bad in "To Watch"

INSERT INTO "Watched" (user_id, movie_id, watch_date)
VALUES
    (1, 1, '2023-11-01'), -- John watched Inception
    (2, 2, '2023-10-15'); -- Jane watched Breaking Bad

INSERT INTO "Rating" (user_id, movie_id, value)
VALUES
    (1, 1, 9),  -- John rate 9/10 Inception
    (2, 1, 9),  -- Jane rate 7/10 Inception
    (2, 2, 10); -- Jane rate 10/10 Breaking Bad

INSERT INTO "Review" (user_id, movie_id, text)
VALUES
    (1, 1, 'Amazing movie, highly recommend!'), -- John reviewed on Inception
    (2, 2, 'Best TV show ever!');               -- Jane reviewed on Breaking
Bad

INSERT INTO "Comment" (user_id, review_id, text)
VALUES
    (2, 1, 'I agree with you!'), -- Jane comment John's review of Inception
    (1, 2, 'Thanks!');           -- John comment Jane's review of Breaking
Bad

INSERT INTO "Report" (user_id, review_id, text, issue)
VALUES
    (3, 1, 'Spam content', 'Spam');
```

6. PL/pgSQL-функции и процедуры для выполнения критически важных запросов

```
CREATE OR REPLACE FUNCTION is_best_seller(arg_movie_id INT)
RETURNS BOOLEAN AS $$
DECLARE
    avg_rating NUMERIC;           -- Средний рейтинг фильма
    watched_count INT;            -- Количество активных пользователей,
которые посмотрели фильм
    watched_threshold NUMERIC;    -- Пороговое значение (10% от активных
пользователей)
BEGIN
    -- 1. Рассчитываем средний рейтинг фильма
    SELECT AVG(value) INTO avg_rating
    FROM "Rating"
    WHERE movie_id = arg_movie_id;
    -- Если нет рейтингов или средний рейтинг ниже 8, возвращаем FALSE
    IF avg_rating IS NULL OR avg_rating < 8 THEN
        RETURN FALSE;
    END IF;

    -- 2. Определяем количество активных пользователей за последний год
    CREATE TEMPORARY VIEW active_users AS
      SELECT user_id AS id
      FROM "Watched"
      WHERE watch_date >= CURRENT_DATE - INTERVAL '1 year'
      GROUP BY user_id
      HAVING COUNT(movie_id) > 10;

    -- Если активных пользователей нет, возвращаем FALSE
    IF ((SELECT COUNT(DISTINCT id) FROM active_users) = 0) THEN
        RETURN FALSE;
    END IF;

    -- 3. Считаем, сколько активных пользователей посмотрело данный фильм
    SELECT COUNT(DISTINCT user_id) INTO watched_count
    FROM "Watched"
    WHERE movie_id = arg_movie_id AND user_id IN (SELECT id FROM
"active_users);

    -- 4. Определяем, соответствует ли количество просмотров порогу (10% от
активных пользователей)
    watched_threshold := active_users * 0.1;

    IF watched_count >= watched_threshold THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

7. Индексы

```
-- Изначально создаем приватные списки к просмотру
CREATE INDEX idx_watchlist_visibility ON "WatchList"(visibility) WHERE
visibility IS FALSE;

-- Очень часто читаем, редко пишем
CREATE INDEX idx_movie_title ON "Movie" USING GIN(title);
CREATE INDEX idx_movie_production_country ON "Movie"(production_country);
CREATE INDEX idx_movie_genres ON "Movie" USING GIN(genres);
CREATE INDEX idx_movie_tags ON "Movie" USING GIN(tags);
CREATE INDEX idx_movie_director ON "Movie"(director);
CREATE INDEX idx_movie_release_date ON "Movie"(release_date);

-- Показываем ревью как на странице пользователя, так и на странице фильма
CREATE INDEX idx_review_visible ON "Review"(visible) WHERE visibility IS
FALSE;

-- Показываем ревью на странице фильма
CREATE INDEX idx_comment_visible ON "Comment"(visible) WHERE visibility IS
FALSE;

-- При смене пароля ищем "PasswordChange" по токену из ссылки
CREATE INDEX idx_passwordchange_token ON "PasswordChange"(token);
CREATE INDEX idx_passwordchange_date ON "PasswordChange"(date);
```

8. Вывод

В результате выполнения второго этапа курсовой работы была разработана полноценная реляционная база данных на основе предварительно сформулированной ER- и даталогической модели базы данных для будущей информационной системы. При реализации функции и создания индексов в реляционной БД PostgreSQL, мы несколько раз пересмотрели даталогическую модель, добавив недостающие поля и ограничения. На основе проделанной работы была создана структура базы данных, которая отвечает поставленным требованиям, гарантирует необходимую целостность хранения данных. Вероятно, реализованная система может быть переработана в дальнейшем с учетом новых требований.