САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет: Программной инженерии и компьютерной техники
Кафедра: Информатики и прикладной математики
Направление подготовки: 09.03.04 (Программная инженерия)

Дисциплина «Базы Данных»

Курсовая работа

Группа: P3117

Работу выполнили
Минин Александр
Молодецкий Арсений

2018г.

Предметная область

(оригинальная версия от заказчика с заметками)

В качестве предметной области будет рассмотрена система кинотеатров в городе Санкт-Петербург. База данных будет представлять собой некоторое состояние кинотеатров в определенный период времени. Целью такого описания является составление бд, по которой бы было удобно определять наиболее релевантные варианты похода в кино, которые можно оформить в виде информационного ресурса. Мы хотим создать ресурс, помогающий пользователям выбрать интересующий их фильм. По актуальности можно отметить, что это одно из наиболее посещаемых видов развлечений под крышей, особенно в таком пасмурном городе.
Как всем известно, в кинотеатрах показывают фильмы во время сеансов в кинозалах. . Фильмы это продукты киноискусства, которое зародилось недавно. Каждая кинокартина имеет длительность показа (иначе говоря, продолжительность сеанса) и жанр (чаще всего фильм относится сразу к нескольким). Стоит также отметить, что кинолента имеет год выпуска и место съемки. Современная публика достаточно требовательная, поэтому ей необходимо больше разнообразия пейзажа на экране. Для этого кадры снимают в разных частях света, разные студии на множестве различных съёмочных площадок. В съемках также принимают участие различные актеры, операторы, сценаристы, режиссеры, продюсеры, композиторы, художники, монтажеры и многие другие (съемочная группа)[1]. Аренда съемочной площадки, перелеты группы да и сама оплата работы всего персонала требует колоссального количества денежных средств. Эти деньги должны укладываться в бюджет фильма (в теории). В конце работы над фильмом в ход идут маркетологи. Они придумывают броский слоган, призывающий людей посмотреть фильм (желательно в месте проката или на DVD), тем самым подогревая интерес зрителей и повышая рейтинг ожидаемости. Ко всему прочему, данные специалисты назначают даты премьеры и показа, продумывают разнообразный медиа контент (саундтреки, постеры, афиши, трейлеры, рекламу). Чаще всего слоган отражает возрастной рейтинг кинокартины (G — General audiences, PG — Parental guidance suggested, PG-13 — Parents strongly cautioned, R — Restricted, NC-17 — No One 17 & Under Admitted), но назначается он в зависимости от содержания фильма. После показа кинокритики выставляют рейтинги фильму и пишут отзывы. В случае приложения на смартфоне 1 пользователь сможет поставить оценку от 1 до 10 для каждого фильма 1 раз, а в случае сайта можно будет выставлять оценку неограниченное количество раз и по этим оценкам тоже выстраивается рейтинг фильмов[2]. ~~После того, как фильм закончился в прокате, появляются данные о кассовых сборах по всему миру и происходит релиз на DVD и Blu-ray.~~ Также фильмы могут быть номинированы или премированы на разных церемониях всякими наградами, что тоже может быть интересно зрителю. И наконец если фильм понравился пользователю, он может посмотреть список похожих фильмов (схожесть по жанрам, оценкам, съёмочной группе, студии и т.д.).
Теперь рассмотрим систему кинотеатров. Она представляет собой совокупность всех сетей и частных кинотеатров (кинотеатры, которые не входят в сети кинотеатров), находящихся в черте города. У каждого кинотеатра есть физический адрес, список сеансов в календаре событий. Также в кинотеатрах есть залы, в которых проходят сеансы, во время которых показываются определенные фильмы в определенные дату и время. Сеансы имеют длительность и цену билета, которая зачастую зависит от времени суток, дня недели. а также расположение места в кинозале[3]. Места в кинозале пронумерованы и сгруппированы по рядам, что позволяет посетителю без

особого труда найти подходящее место или посмотреть занятость зала, ведь места могут быть свободны, куплены или забронированы.
Таким образом, благодаря нашей системе, можно, не теряя времени, узнать информацию об интересующем нас фильме, а также найти удобное место для просмотра киноленты. Хотим отметить, что наша система может способствовать привлечению потока клиентов в кинотеатры, так как пользователи смогут оптимально выбирать релевантный сеанс.

В ходе обсуждения проекта с заказчиком были уточнены детали:
1) Над одним фильмом может работать несколько групп (команд), состоящих из работников киноиндустрии: режиссера, оператора и так далее. Каждый человек в группе имеет одну или более должностей (ролей).
2) Данные о релизах хранить не нужно. Пользователь сможет выставить оценку для фильма только один раз: и на смартфоне, и на сайте.
3) Следует хранить расположение мест в зале и “базовую” стоимость (тариф) для каждого места, которую можно менять для каждого отдельно взятого билета/сеанса. В терминах инфологической модели БД, у нас должны быть атрибуты “стоимость” и в таблице, предназначенной для хранения билетов, и в таблице, предназначенной для хранения мест и их конфигурации.

Даталогическая модель

```
CREATE TABLE "Сети" (
	"ид" SERIAL NOT NULL,
	"название" TEXT NOT NULL UNIQUE,
	"сайт" TEXT UNIQUE,
	PRIMARY KEY ("ид")
);

CREATE TABLE "Кинотеатры" (
	"ид" SERIAL NOT NULL,
	"ид_сети" INTEGER NOT NULL REFERENCES "Сети" ON DELETE CASCADE,
	"название" TEXT NOT NULL,
	"город" TEXT NOT NULL,
	"адрес" TEXT NOT NULL,
	PRIMARY KEY ("ид")
);

CREATE TABLE "Залы" (
	"ид" SERIAL NOT NULL,
	"ид_кинотеатра" INTEGER NOT NULL REFERENCES "Кинотеатры" ON DELETE
CASCADE,
	"номер_зала" INTEGER NOT NULL UNIQUE CONSTRAINT csr_room_id CHECK
("номер_зала" > 0),
	PRIMARY KEY ("ид")
);

CREATE TABLE "Места" (
	"ид" SERIAL NOT NULL,
	"ид_зала" INTEGER NOT NULL REFERENCES "Залы" ON DELETE CASCADE,
	"ряд" INTEGER NOT NULL
		CONSTRAINT csr_row CHECK ("ряд" > 0),
	"место" INTEGER NOT NULL
		CONSTRAINT csr_place CHECK ("место" > 0),
	"стоимость" INTEGER NOT NULL
		CONSTRAINT csr_seat_price CHECK ("стоимость" > 0),
	CONSTRAINT csr_unique_seat UNIQUE (ид_зала, ряд, место),
	PRIMARY KEY ("ид")
);

CREATE TABLE "Жанры" (
	"ид" SERIAL NOT NULL,
	"название" TEXT NOT NULL UNIQUE,
	PRIMARY KEY ("ид")
);


CREATE TABLE "Люди" (
	"ид" SERIAL NOT NULL,
	"фио" TEXT NOT NULL,
	PRIMARY KEY ("ид")
```

```
);

CREATE TABLE "Пользователи" (
	"ид" SERIAL NOT NULL,
	"логин" TEXT NOT NULL UNIQUE,
	"пароль" TEXT NOT NULL,
	"фио" TEXT NOT NULL,
	PRIMARY KEY ("ид")
);

CREATE TABLE "Фильмы" (
	"ид" SERIAL NOT NULL,
	"название" TEXT NOT NULL,
	"начало_съемок" TIMESTAMP NOT NULL,
	"конец_съемок" TIMESTAMP,
	"премьера" TIMESTAMP,
	"продолжительность" INTEGER NOT NULL
		CONSTRAINT csr_duration CHECK ("продолжительность" > 0),
	"бюджет" int,
		CONSTRAINT csr_budget CHECK ("бюджет" > 0),
	"возрастной_рейтинг" VARCHAR(5) NOT NULL
		CONSTRAINT csr_age_rate CHECK("возрастной_рейтинг"
in('G','PG','PG-13','R','NC-17')),
	"слоган" TEXT,
	"кассовые_сборы" int,
		CONSTRAINT csr_money CHECK ("кассовые_сборы" > 0),
	CONSTRAINT csr_movie_start_end_range CHECK ("начало_съемок" <
"конец_съемок"),
	CONSTRAINT csr_movie_release_end CHECK ("конец_съемок" < "премьера"),
	PRIMARY KEY ("ид")
);


CREATE TABLE "Медиа" (
	"ид" SERIAL NOT NULL,
	"название" TEXT NOT NULL,
	"ид_фильма" INTEGER NOT NULL REFERENCES "Фильмы" ON DELETE
CASCADE,
	"тип" TEXT NOT NULL,
	"url" TEXT NOT NULL,
	PRIMARY KEY ("ид")
);

CREATE TABLE "Оценки" (
	"ид" SERIAL NOT NULL,
	"ид_фильма" INTEGER NOT NULL REFERENCES "Фильмы" ON DELETE
CASCADE,
	"ид_пользователя" INTEGER NOT NULL REFERENCES "Пользователи" ON
DELETE CASCADE,
	"значение" INTEGER NOT NULL CONSTRAINT csr_rate CHECK ("значение"
BETWEEN 1 AND 10),
	"комментарий" TEXT,
```

```
        "дата_время" TIMESTAMP NOT NULL,
        PRIMARY KEY ("ид")
);

CREATE TABLE "Награды" (
        "ид" SERIAL NOT NULL,
        "ид_фильма" INTEGER NOT NULL REFERENCES "Фильмы" ON DELETE
CASCADE,
        "ид_человека" INTEGER NOT NULL REFERENCES "Люди" ON DELETE CASCADE,
        "название" TEXT NOT NULL,
        "тип" TEXT NOT NULL,
        "дата" TIMESTAMP NOT NULL,
        PRIMARY KEY ("ид")
);

CREATE TABLE "Группы" (
        "ид" SERIAL NOT NULL,
        "название" TEXT NOT NULL,
        PRIMARY KEY ("ид")
);

CREATE TABLE "Роли" (
        "название" TEXT NOT NULL,
        "ид_человека" INTEGER NOT NULL REFERENCES "Люди" ON DELETE CASCADE,
        "ид_группы" INTEGER NOT NULL REFERENCES "Группы" ON DELETE CASCADE,
        PRIMARY KEY ("название","ид_человека","ид_группы")
);


CREATE TABLE "Сеансы" (
        "ид" SERIAL NOT NULL,
        "ид_фильма" INTEGER NOT NULL REFERENCES "Фильмы" ON DELETE
CASCADE,
        "ид_зала" INTEGER NOT NULL REFERENCES "Залы" ON DELETE CASCADE,
        "начало" TIMESTAMP NOT NULL,
        "конец" TIMESTAMP NOT NULL,
        CONSTRAINT csr_session_date CHECK ("начало" < "конец"),
        PRIMARY KEY ("ид")
);


CREATE TABLE "Билеты" (
        "ид" SERIAL NOT NULL,
        "ид_сеанса" INTEGER NOT NULL REFERENCES "Сеансы" ON DELETE CASCADE,
        "ид_места" INTEGER NOT NULL REFERENCES "Места" ON DELETE RESTRICT,
        "ид_пользователя" INTEGER REFERENCES "Пользователи" ON DELETE
RESTRICT,
        "стоимость" INTEGER NOT NULL CONSTRAINT csr_ticket_price CHECK
("стоимость" > 0),
        "статус" INTEGER NOT NULL,
                CONSTRAINT csr_tickets_state CHECK("статус" BETWEEN 0 AND 2),
        CONSTRAINT csr_one_ticket_per_seat UNIQUE (ид_сеанса, ид_места),
```

```
	PRIMARY KEY ("ид")
);


CREATE TABLE "Фильмы_Жанры" (
	"ид_фильма" INTEGER NOT NULL REFERENCES "Фильмы" ON DELETE
CASCADE,
	"ид_жанра" INTEGER NOT NULL REFERENCES "Жанры" ON DELETE CASCADE,
	PRIMARY KEY ("ид_фильма", "ид_жанра")
);

CREATE TABLE "Фильмы_Группы" (
	"ид_фильма" INTEGER NOT NULL REFERENCES "Фильмы" ON DELETE
CASCADE,
	"ид_группы" INTEGER NOT NULL REFERENCES "Группы" ON DELETE CASCADE,
	PRIMARY KEY ("ид_фильма", "ид_группы")
);
```

Триггеры

```
CREATE OR REPLACE FUNCTION проверка_сеанса() RETURNS trigger AS $$
DECLARE
	премьера timestamp;
BEGIN
SELECT Фильмы.премьера INTO премьера FROM "Фильмы" WHERE ид =
NEW.ид_фильма;
IF NEW.начало < премьера THEN
	RAISE EXCEPTION 'Дата премьеры фильма (%) не может быть позже, чем дата
начала показа (%)', премьера, NEW.начало;
END IF;

IF EXISTS (SELECT 1 FROM Сеансы WHERE ид != NEW.ид AND ид_зала =
NEW.ид_зала AND
	(
		(NEW.конец >= Сеансы.начало AND NEW.конец <= Сеансы.конец) OR
		(NEW.начало >= Сеансы.начало AND NEW.начало <= Сеансы.конец)
	)
) THEN
	RAISE EXCEPTION 'В одном зале не могут одновременно проходить два сеанса';
END IF;

RETURN NEW;
END;
$$  LANGUAGE plpgsql;

CREATE TRIGGER "проверка_сеанса" BEFORE INSERT OR UPDATE ON "Сеансы"
FOR EACH ROW EXECUTE PROCEDURE проверка_сеанса();
----

CREATE OR REPLACE FUNCTION награды_до_премьеры_запрещены() RETURNS
trigger AS $$
```

```
DECLARE
	премьера timestamp;
BEGIN
SELECT Фильмы.премьера INTO премьера FROM "Фильмы" WHERE ид =
NEW.ид_фильма;
IF now() < премьера THEN
	RAISE EXCEPTION 'Награда не может вручаться до выхода фильма (премьера
%)', премьера;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER "награды_до_премьеры_запрещены" BEFORE INSERT OR UPDATE
ON "Награды"
FOR EACH ROW EXECUTE PROCEDURE награды_до_премьеры_запрещены();
----

CREATE OR REPLACE FUNCTION оценки_до_премьеры_запрещены() RETURNS trigger
AS $$
DECLARE
	премьера timestamp;
BEGIN
SELECT Фильмы.премьера INTO премьера FROM "Фильмы" WHERE ид =
NEW.ид_фильма;
IF NEW.дата_время < премьера THEN
	RAISE EXCEPTION 'Оценка не может быть поставлена до премьеры фильма
(премьера %)', премьера;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER "оценки_до_премьеры_запрещены" BEFORE INSERT OR UPDATE
ON "Оценки"
FOR EACH ROW EXECUTE PROCEDURE оценки_до_премьеры_запрещены();

----

CREATE OR REPLACE FUNCTION проверка_билета() RETURNS trigger AS $$
DECLARE
	места_ид_зала int;
	сеансы_ид_зала int;
BEGIN
SELECT Места.ид_зала INTO места_ид_зала FROM "Места" WHERE ид =
NEW.ид_места;
SELECT Сеансы.ид_зала INTO сеансы_ид_зала FROM "Сеансы" WHERE ид =
NEW.ид_сеанса;
IF места_ид_зала <> сеансы_ид_зала THEN
	RAISE EXCEPTION 'ИД зала, указанный в информации о месте билета (%) не
соответствует ИД зала, указанному в сеансах (%)', места_ид_зала, сеансы_ид_зала;
END IF;
```

```
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER "проверка_билета" BEFORE INSERT OR UPDATE ON "Билеты"
FOR EACH ROW EXECUTE PROCEDURE проверка_билета();
```

Типичные запросы

Типичные (распространенные) запросы к СУБД составлены на основании предметной области и предполагаемых моделей использования БД. Здесь приведены самые распространенные запросы, которые позволяют нам понять, какие индексы нужно ввести в первую очередь, чтобы СУБД как можно более эффективно справлялась с выполнением требований заказчика.

- Для вывода оценок за фильмы у пользователя и любимых жанров

```
SELECT * FROM Оценки JOIN Фильмы ON Фильмы.ид = Оценки.ид_фильма WHERE
Оценки.ид_пользователя = ?;
SELECT * FROM Оценки JOIN Фильмы_Жанры ON Фильмы_Жанры.ид_фильма =
Оценки.ид_фильма;
```

- Для вывода оценок за фильм

```
SELECT * FROM Оценки JOIN Фильмы ON Фильмы.ид = Оценки.ид_фильма WHERE
Оценки.ид_фильма = ?;
```

- Для сравнения вкусов к фильмам у двух пользователей

```
SELECT Фильмы.название, Оценки.значение AS Оценка_1ого, t.значение AS
Оценка_2ого
FROM Оценки
JOIN Фильмы ON Фильмы.ид = Оценки.ид_фильма
JOIN Оценки AS t ON t.ид_фильма = Оценки.ид_фильма AND t.ид_пользователя <>
Оценки.ид_пользователя
WHERE Оценки.ид_пользователя = ? AND t.ид_пользователя = ?;
```

- Для авторизации

```
SELECT * FROM Пользователи WHERE логин = ?;
```

- Для вывода информации сеансах на фильмы, идущие в данный момент

```
SELECT * FROM Фильмы
JOIN Сеансы
WHERE Фильмы.ИД = Сеансы.ИД Дата_время_начала #(или дата_время_Конца)# < ?
```

- Для вывода информации о конкретном сеансе и билетах

```
SELECT * FROM Сеансы
JOIN Залы ON Сеансы.ид_зала = Залы.ид
JOIN Места ON Места.ид_зала = Залы.ид
LEFT JOIN Билеты ON Билеты.ид_сеанса = Сеансы.ид AND Билеты.ид_места =
Места.ид
WHERE Сеансы.ид = ?;
```

Индексы

```
CREATE INDEX Оценки_пользователь_ind ON Оценки (ид_пользователя);
CREATE INDEX Сеансы_дата_начала_ind ON Сеансы(дата_начала);
CREATE INDEX Сеансы_дата_конца_ind ON Сеансы(дата_конца);
CREATE INDEX Пользователи_логин_ind ON Пользователи(логин);
```