НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
(повна назва інституту/факультету)

КАФЕДРА інформатики та програмної інженерії
(повна назва кафедри)

КУРСОВА РОБОТА

з дисципліни «Бази даних»
(назва дисципліни)

на тему: база даних для підтримки діяльності кіностудії

Студента __2__ курсу _ІП-13 групи
спеціальності 121 «Інженерія програмного забезпечення»

Дойчева К.М.
(прізвище та ініціали)

Керівник Ліщук О.В.
_____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна
шкала_____

Кількість балів: _____ Оцінка ECTS____

Члени комісії _____. _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2022 рік

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет Інформатики та обчислювальної техніки

Кафедра Інформатики та програмної інженерії

Дисципліна Бази даних

Курс 2 Група ІП-13 Семестр 1

ЗАВДАННЯ

на курсову роботу студента

Дойчева Костянтина Миколайовича

(прізвище, ім'я, по батькові)

1. Тема роботи База даних для підтримки діяльності кіностудії

керівник роботи Ліщук О.В.

2. Строк здачі студентом закінченої роботи 16.01.2023

3. Вихідні дані до роботи створена база даних відповідно до умови, SQL скрипти

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Опис предметного середовища, постановка задачі, побудова ER-діаграми, побудова Реляційної моделі бази даних, реалізація бази даних, створення користувачів, SQL запити

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
ER-діаграма, реляційна схема бази даних, приклади використання SQL скриптів

6. Дата видачі завдання 31.10.2022

КАЛЕНДАРНИЙ ПЛАН

/п	Назва етапів виконання курсового проекту	Строк виконання етапів проекту	Примітка
1.	Вступ	31.10.2021-01.11.2021	
2.	Опис предметного середовища	01.11.2021-03.11.2021	
3.	Постановка задачі	01.11.2021-03.11.2021	
4.	Побудова ER-моделі	03.11.2021-13.11.2021	
5.	Побудова реляційної схеми	13.11.2021-17.11.2021	
6.	Створення бази даних з допомогою обраної СУБД	17.11.2021-19.11.2021	
7.	Імпортування даних	17.11.2021-19.11.2021	
8.	Створення користувачів та реалізація їх функціоналу	19.11.2021-24.11.2021	
9.	Створення SQL запитів	19.11.2021-13.12.2021	
10.	Висновок	13.12.2021-14.12.2021	
11.	Перелік посилань	31.10.2021-14.12.2021	

Студент ____**Дойчев К.М.**
(підпис) (прізвище та ініціали)

Керівник роботи _____**Ліщук О.В.**_____ _____
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Бази даних використовуються для збереження, обробки та доступу до будь-яких даних. Бази даних – це свого роду організоване та оптимізоване сховище інформації.

Бізнеси використовують бази даних для зберігання та маніпуляції з даними. Також вони необхідні для оптимізації та автоматизації бізнес-процесів. Компанії збирають дані про такі бізнес-процеси, як продаж, обробка замовлень, та клієнтські послуги.

У цій роботі представлена база даних для підтримки діяльності кіностудії

1. Опис предметного середовища

Кінотеатр - заклад розваг для людей, де вони можуть насолоджуватися переглядом кіно та іншими сервісами, які надає заклад. Кінотеатри часто мають декілька кінозалів, що дозволяє показувати різні фільми одночасно. Зазвичай кінотеатри розташовані у великих містах чи торгових центрах. Кінотеатри можуть бути різного розміру та спеціалізації - від маленьких незалежних кінотеатрів до великих сітевих кінотеатрів, які пропонують додаткові сервіси, такі як кінотеатральна продажа і сніданки. Кінотеатри також можуть пропонувати різні види квитків, такі як преміум-квитки, які надають додаткові привілеї.

Предметне середовище для кінотеатру містить різні об'єкти та сутності, які пов'язані з функціонуванням такого закладу. Основні об'єкти, які можуть бути включені до такого середовища:

- Кінозали - простір, де проводиться показ фільмів. Кінозали можуть бути різної величини та мати різну кількість місць.
- Фільми - це відеоролики, які показуються в кінотеатрі. Фільми можуть бути різних жанрів та напрямків (комедії, драми, фантастика тощо).
- Каси - місця, де купуються квитки на фільм. Каси можуть бути розташовані в різних місцях кінотеатру (наприклад, у вході або у коридорі).
- Бари/магазини - місця, де можна придбати додаткові товари для поліпшення вражень (напої, снеки …)

2. Постановка задачі

Метою даної роботи є розробка бази даних для підтримки діяльності кіностудії. Результатом проектування повинна бути БД з визначеною структурою, заповнена даними та оптимізована для потреб користувача. Необхідно створити об’єкти, які покращать роботу розробника та користувача.

Завданням курсової роботи є розробка бази даних і її використання для вирішення практичних задач.

При розробці бази даних необхідно враховувати:

- вимоги до функціональності (наявність усіх функцій, які необхідні для реалізації поставленої задачі);
- вимоги до цілісності даних;
- вимоги до мінімізації об’єму даних, що зберігаються;
- наявність багатокористувальницького режиму;
- вимоги до швидкодії.

В процесі роботи над курсовою роботою повинні бути виконані наступні завдання:

- побудувати ER-модель, для чого необхідно:
    - детально проаналізувати предметне середовище;
    - сформулювати бізнес-правила, які будуть основою завдання обмежень при проектуванні та реалізації бази даних;
    - виявити необхідний набір сутностей;
    - визначити необхідний набір атрибутів для кожної сутності;
    - визначити зв'язки між об'єктами;
    - описати отриману ER-модель в одній з відомих нотацій;
    - розробити модель користувачів бази даних з описом їх прав;
- побудувати реляційну схему з ER-моделі, для чого необхідно:
    - побудувати набір необхідних відношень бази даних;
    - виділити первинні і зовнішні ключі у кожному з відношень;
    - привести отримані відношення до третьої нормальної формі;
    - визначити обмеження цілісності для спроектованих відношень;

- створити базу даних, що була спроектована, у форматі обраної системи управління базою даних (СУБД);
- створити користувачів бази даних, реалізувавши розроблену багатокористувальницьку модель;
- імпортувати дані з використанням засобів СУБД в створену базу даних;
- мовою SQL написати запити для визначених на етапі аналізу предметного середовища потреб користувачів;
- оптимізувати роботу запитів (продемонструвати роботу до і після оптимізації).

3. ER-діаграма

Після аналізу було виділено такі сутності та зв’язки між ними:

3.1 Бізнес-правила

1) Сеанс (event) для кінофільму не може бути раніше ніж вихід (release_date) кінокартини
2) Мережа кінотеатрів може мати декілька кінотеатрів за різними адресами
3) Кінозала може мати різні за типом сидіння, які відрізнятимуться в ціні
    a. Сидіння заднього ряду вважаються більш преміальними
    b. Деякі кінозали можуть мати сидіння-ліжка
4) Окрім квитків кінотеатр має змогу продавати товари, які можуть знадобитися відвідувачам під час сеансу
    a. 3D-окуляри
    b. Напої та їжа
    c. Засоби гігієни

3.2 Вибір сутностей

- Кінотеатр (Cinema)
- Адреса (Address)
- Кінозала (Room)
- Місце для сидіння (Seat)
- Фільм (Movie)
- Ціна на квиток до фільму (Price)
- Подія (Event)
- Квиток (Ticket)
- Товар (Product)
- Замовлення (Order)
- Елемент замовлення (OrderItem)
- Відвідувач (Customer)

3.3 Набори атрибутів сутностей

Таблиця 3.1 – Сутності та їхні атрибути

Сутність	**Атрибути**
Cinema	id, name, address_id, phone_number, email
Address	id, country, city, street, house_number, index
Room	id, name, cinema_id
Seat	id, room_id, row, number, type
Movie	id, title, description, duration, release_date, rating, genres, directors, actors, poster_url
Price	id, movie_id, seat_type, price, currency
Event	id, movie_id, room_id, starts_at, ends_at
Ticket	id, event_id, seat_id, price
Product	id, name, price
Order	id, customer_id, price
OrderItem	id, order_id, ticket_id, product_id, quantity
Customer	id, first_name, last_name, phone_number, email, password

Сутність **Cinema** - кінотеатр мережі. Вона включає в себе набір кімнат (**Room**) і, відповідно, місць в цих кімнатах (**Seat**). Також кожен кінотеатр має адресу (**Address**).

Address - сутність адресу кінотеатру, яка має дані стосовно його географічного положення

Room - кімната конкретного кінотеатру. **Room** пов’язаний з кінотеатром за допомогою many-to-to relation атрибута *address_id*.

Seat - місце конкретної кімнати. Many-to-one relation з **Room** досягається завдяки атрибуту *room_id*. Також варто звернути увагу на атрибут *type*, який визначатиме тип місця. Наприклад: standard, premium, lux.

Movie - кінофільм. Містить детальну інформацію про кінокартину. Поле duration - тривалість фільму у секундах. У полі *poster_url* зберігатиметься посилання на постер зі стороннього ресурсу.

Price - ціна на обраний тип місця для обраного фільму. Атрибут *movie_id* визначає many-to-one relation з **Movie**, *seat_type* - має ті ж значення, що атрибут *type* в **Seat**, price - ціна за квиток в копійках, currency - валюта за стандартом ISO 4217.

Event - подія. Кожна подія має обов’язково мати фільм та кінозал, цей зв’язок надають атрибути *movie_id* (**Movie**) та *room_id* (**Room**), які є one-to-one mandatory relation. Атрибути starts_at та ends_at - дати початку та кінця в форматі timestamp.

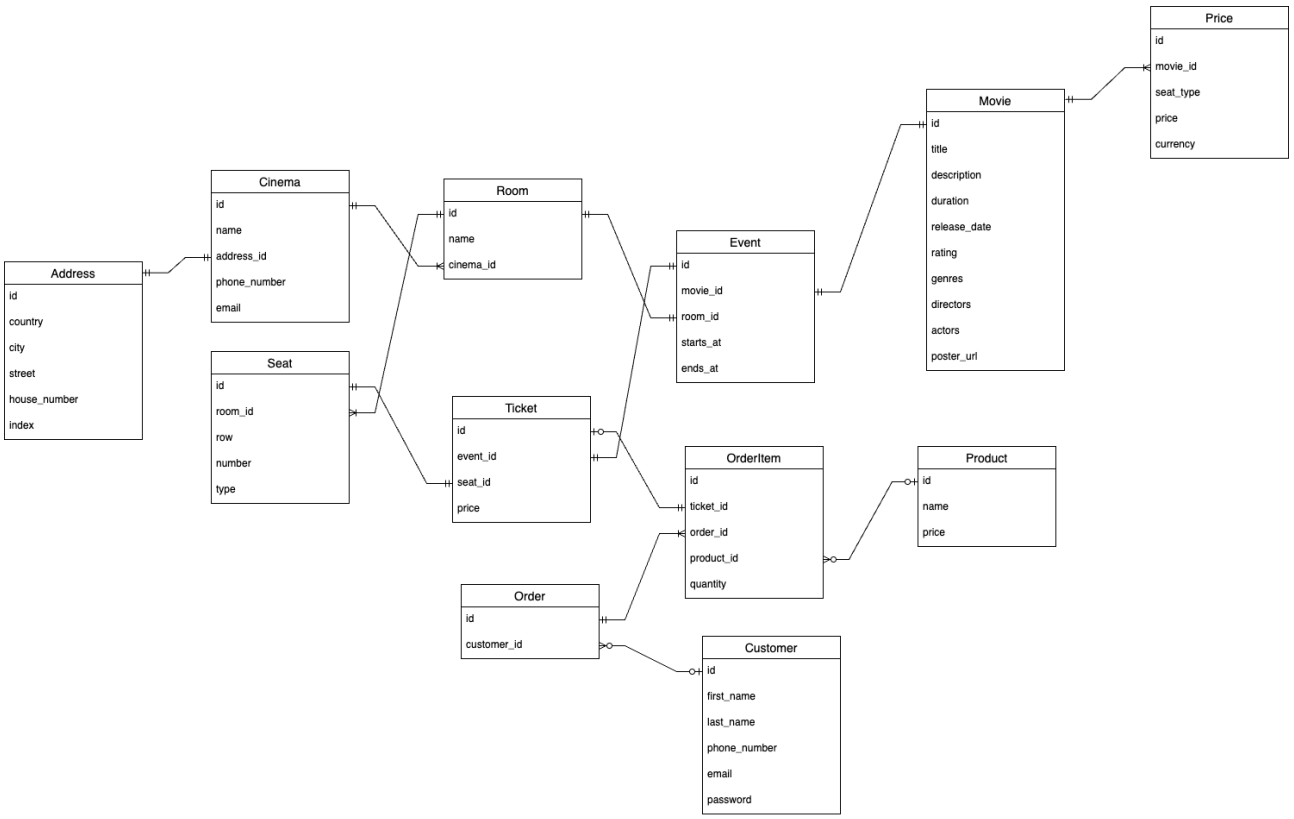
Ticket - квиток на подію. Він обов’язково має зв’язок з **Event** та **Seat** використовуючи атрибути *event_id* та *seat_id* відповідно. Атрибут *price* - ціна за квиток в копійках.

Product - товар, який відвідувач може придбати для покращення вражень від перебування. Атрибут *price* - ціна в копійках.

Order - замовлення відвідувача. У разі, якщо відвідувач зробив замовлення з свого аккаунту, у *customer_id* зберігається індивідуальний номер користувача (many-to-one optional relation). У разі замовлення на касі, це поле буде null.

OrderItem - елемент замовлення. Це може бути як квиток (one-to-one optional relation) *ticket_id*, так і товар (many-to-one optional relation) *product_id*. Атрибут *order_id* має обов'язково мати ідентифікатор замовлення (**Order**). Атрибут *quantity* відповідає за кількість обраного товару

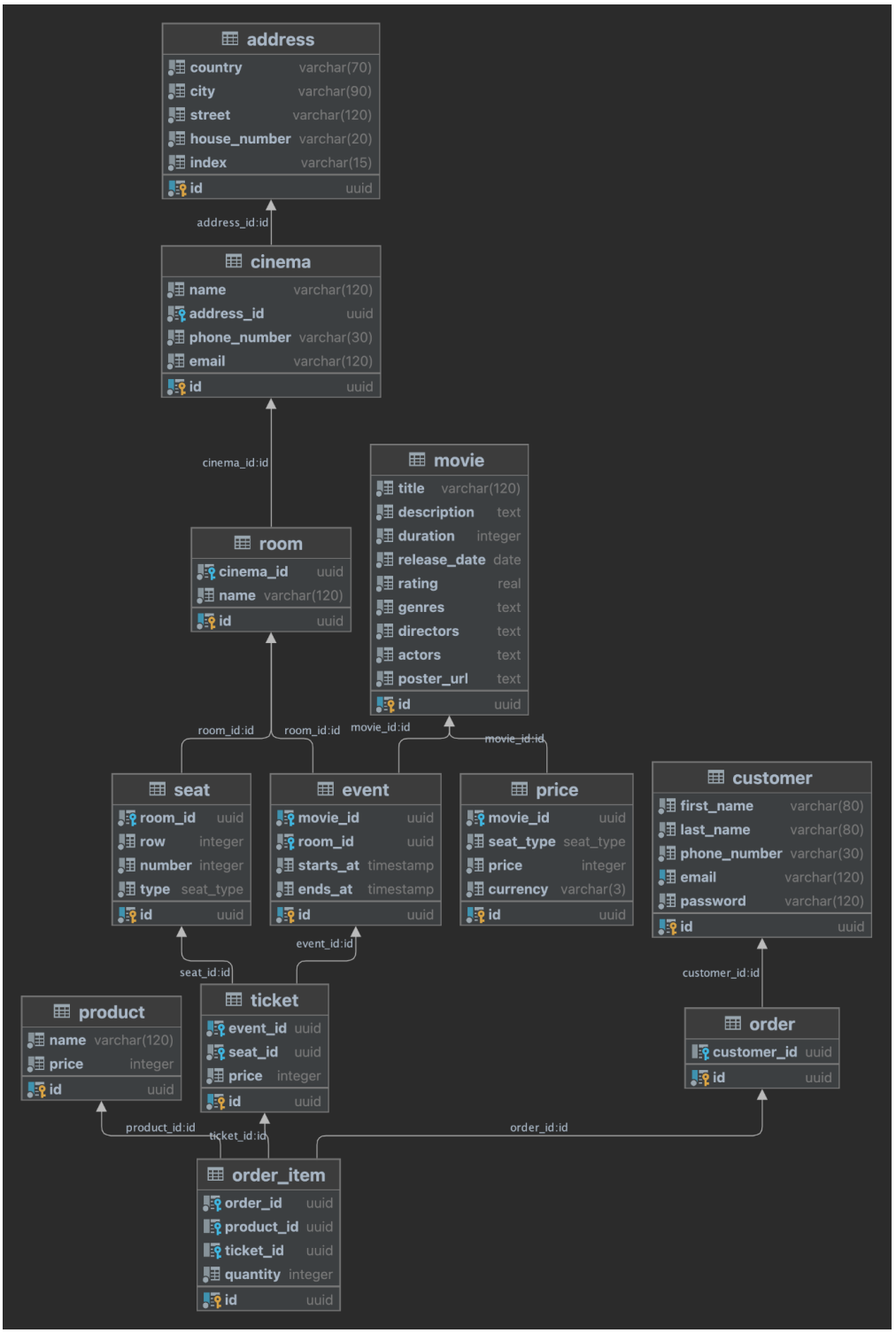
Customer - відвідувач кінотеатру. Має атрибути з персональними даними. Атрибут *email* має бути унікальним для кожного користувача.



Зображення 3.1 – ER діаграма

4. Реляційна модель бази даних

4.1 Побудова необхідних відношень та визначення первинних і зовнішніх ключів



Зображення 4.1 – Реляційна схема бази даних

На схемі (Зображення 4.1) видно, що база даних знаходиться у 3 нормальній формі, тому що всі поля таблиць декомпозовані а також всі

атрибути таблиць мають функціонально повну залежність від первинного ключа (primary key), кожен неключовий атрибут не є транзитивно залежним від первинного ключа.

1. Визначення обмежень цілісності для спроектованих відношень. Обмеження цілісності: Рядок батьківської таблиці може бути видалений лише у тому випадку, якщо немає зовнішніх ключів, що посилаються на значення референційного (reference) ключа цього рядка. (Реалізовується відсутністю параметра ON DELETE CASCADE)
2. Обов’язкові атрибути таблиць мають обмеження NOT NULL, для запобіганню помилок при роботі з даними і визначення важливості полів.

5. Реалізація бази даних

5.1 Ініціалізація таблиць

```
create type "public"."seat_type" as enum ('standard', 'premium', 'luxury');

create table if not exists "address" (
    "id" uuid primary key default gen_random_uuid(),
    "country" varchar(70) not null,
    "city" varchar(90) not null,
    "street" varchar(120) not null,
    "house_number" varchar(20) not null,
    "index" varchar(15) not null
);

create table if not exists "cinema" (
    "id" uuid primary key default gen_random_uuid(),
    "name" varchar(120) not null,
    "address_id" uuid not null,
    "phone_number" varchar(30) not null,
    "email" varchar(120) not null
);

alter table "cinema" add constraint "fk_cinema_address" foreign key
("address_id") references "address" ("id");

create table if not exists "room" (
    "id" uuid primary key default gen_random_uuid(),
    "cinema_id" uuid not null,
    "name" varchar(120) not null
);

alter table "room" add constraint "fk_room_cinema" foreign key ("cinema_id")
references "cinema" ("id");

create table if not exists "seat" (
    "id" uuid primary key default gen_random_uuid(),
    "room_id" uuid not null,
    "row" int not null,
    "number" int not null,
    "type" seat_type not null
);

alter table "seat" add constraint "fk_seat_room" foreign key ("room_id")
references "room" ("id");

create table if not exists "movie" (
```

```
    "id" uuid primary key default gen_random_uuid(),
    "title" varchar(120) not null,
    "description" text not null,
    "duration" int not null,
    "release_date" date not null,
    "rating" real not null,
    "genres" text not null,
    "directors" text not null,
    "actors" text not null,
    "poster_url" text not null
);

create table if not exists "price" (
    "id" uuid primary key default gen_random_uuid(),
    "movie_id" uuid not null,
    "seat_type" seat_type not null,
    "price" int not null,
    -- ISO 4217 currency code
    "currency" varchar(3) not null
);

alter table "price" add constraint "fk_price_movie" foreign key ("movie_id")
references "movie" ("id");

create table if not exists "event" (
    "id" uuid primary key default gen_random_uuid(),
    "movie_id" uuid not null,
    "room_id" uuid not null,
    "starts_at" timestamp not null,
    "ends_at" timestamp not null
);

alter table "event" add constraint "fk_event_movie" foreign key ("movie_id")
references "movie" ("id");
alter table "event" add constraint "fk_event_room" foreign key ("room_id")
references "room" ("id");

create table if not exists "ticket" (
    "id" uuid primary key default gen_random_uuid(),
    "event_id" uuid not null,
    "seat_id" uuid not null,
    "price" int not null
);

alter table "ticket" add constraint "fk_ticket_event" foreign key
("event_id") references "event" ("id");
alter table "ticket" add constraint "fk_ticket_seat" foreign key ("seat_id")
references "seat" ("id");
```

```
create table if not exists "product" (
    "id" uuid primary key default gen_random_uuid(),
    "name" varchar(120) not null,
    "price" int not null
);

create table if not exists "customer" (
    "id" uuid primary key default gen_random_uuid(),
    "first_name" varchar(80) not null,
    "last_name" varchar(80) not null,
    "phone_number" varchar(30) not null,
    "email" varchar(120) not null,
    "password" varchar(120) not null,
    constraint "user_email" unique ("email")
);

create table if not exists "order" (
    "id" uuid primary key default gen_random_uuid(),
    "customer_id" uuid on delete set null
);

alter table "order" add constraint "fk_order_customer" foreign key
("customer_id") references "customer" ("id");

create table if not exists "order_item" (
    "id" uuid primary key default gen_random_uuid(),
    "order_id" uuid not null,
    "product_id" uuid,
    "ticket_id" uuid,
    "quantity" int not null
);

alter table "order_item" add constraint "fk_order_item_order" foreign key
("order_id") references "order" ("id");
alter table "order_item" add constraint "fk_order_item_product" foreign key
("product_id") references "product" ("id");
alter table "order_item" add constraint "fk_order_item_ticket" foreign key
("ticket_id") references "ticket" ("id");
```

5.2 Створення ролей:

```
create role "manager"
with login
encrypted password '977ef10c-938f-4e5a-a4d2-02c21bb4db0f';

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public to
"manager";
GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO "manager";

create role "viewer"
with login
encrypted password '01bd6962-2f39-4a10-ab95-5a66ed9168ad';

GRANT SELECT ON ALL TABLES IN SCHEMA public to "viewer";
GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO "viewer";

select * from information_schema.table_privileges
where grantee = 'manager';

select * from information_schema.table_privileges
where grantee = 'viewer';

select * from pg_roles where rolname not ilike 'pg_%';
```

6. SQL запити

6.1 Запити

```sql
-- get_events_on_date

create or replace function get_events_on_date(
    search_date date
)
returns table (
    event_id uuid,
    movie_id uuid,
    movie_name varchar(120),
    starts_at timestamp
                )
language plpgsql
as
    $$
    begin
        return query
        select
            e.id as event_id,
            e.movie_id,
            m.title as movie_name,
            e.starts_at
        from
            public.event e
        join
            public.movie m
        on
            e.movie_id = m.id
        where
            e.starts_at::date = search_date;
    end;
    $$;


select * from get_events_on_date('2023-01-12');
```

	event_id	movie_id	movie_name	starts_at
1	6842359f-15b7-45e7-b042-300bbaa635f4	4ee6e7da-6c36-4fbe-a05e-9d057d9c725c	Haley - Upton	2023-01-12 14:18:55.438000
2	22ecc2a0-ec07-4fb2-9513-948383c34eb8	0964fa3b-99df-4f52-94d8-e08cc2af1b3d	Mosciski - Gibson	2023-01-12 20:07:16.742000
3	bd56634a-f126-4bce-9923-f9c93f3dcfdc	fd3da65c-425b-47e5-abaf-5468ae24b962	Lockman and Sons	2023-01-12 14:55:36.925000

```sql
-- get_events_today

create or replace function get_events_today()
returns table (
    event_id uuid,
    movie_id uuid,
    movie_name varchar(120),
    starts_at timestamp
               )
language plpgsql
as
    $$
    begin
        return query
        select
            e.id as event_id,
            e.movie_id,
            m.title as movie_name,
            e.starts_at
        from
            public.event e
        join
            public.movie m
        on
            e.movie_id = m.id
        where
            e.starts_at::date = current_date;
    end;
    $$;


select * from get_events_today();
```

```
-- apply discount to the specific ticket

create or replace procedure apply_discount_for_ticket(
    ticket_id uuid, discount int
    )
    language plpgsql
as
$$
begin
    update ticket
    set price = price - (price * discount / 100)
    where id = ticket_id;
end;
$$;
```

```
select id, price from ticket
where id = 'd76b814e-0807-4c45-a98b-758d1d18e793';
```

1 row | Tx: Auto | DDL | CSV

	id	price
1	d76b814e-0807-4c45-a98b-758d1…	5016

```
call apply_discount_for_ticket( ticket_id: 'd76b814e-0807-4c45-a98b-758d1d18e793', discount: 10);
```

```
select id, price from ticket
where id = 'd76b814e-0807-4c45-a98b-758d1d18e793';
```

1 row | Tx: Auto | DDL | CSV

	id	price
1	d76b814e-0807-4c45-a98b-758d1d18e793	4515

```sql
-- order item with price details
create or replace view detailed_order_item as
select o.id, o.order_id, o.quantity, coalesce(p.price, 0) as product_price,
coalesce(t.price, 0) as ticket_price
from public.order_item as "o"
         left join "product" as "p" on "p".id = "o".product_id
         left join "ticket" t on o.ticket_id = t.id;
```

```sql
select * from detailed_order_item
where id = '454331ac-e468-4bdc-98b4-dc9e4061b810';
```

1 row | Tx: Auto | DDL | CSV

		order_id	quantity	product_price	ticket_price
1	31ac-e468-4bdc-98b4-dc9e4…	6044331c-07c8-4034-893c-b809c…	1	0	3066

```sql
-- calculate order total

create or replace function calculate_order_total(ordr_id uuid, discount int
default 0) returns int
    language plpgsql as
$$
declare
    total           int := 0;
    ordr_item       record;
begin
    for ordr_item in select * from detailed_order_item where order_id =
ordr_id
        loop
            total := total + (ordr_item.quantity * ordr_item.product_price) +
(ordr_item.quantity * ordr_item.ticket_price);
        end loop;

    return total - (total * discount / 100);
end;
$$;
```

```
select calculate_order_total( ordr_id: '6044331c-07c8-4034-893c-b809c16e3180');
```

	calculate_order_total
1	639594

```
select calculate_order_total( ordr_id: '6044331c-07c8-4034-893c-b809c16e3180', discount: 10);
```

	calculate_order_total
1	575635

```
-- on cinema details change
create or replace function handle_cinema_details_change()
    returns trigger
    as $$
    begin
        raise notice 'Do not forget to update the website with the new cinema
details';
        return new;
    end;
    $$
    language plpgsql;

create trigger cinema_details_change
    after update or insert or delete or truncate
    on cinema
    for each statement
    execute procedure handle_cinema_details_change();
```

```
cinema.public> update cinema
               set name = 'Renner and Sons updated'
               where id = '3f7451c0-0f0b-4b66-9c62-a7526e625fc4'
Do not forget to update the website with the new cinema details
[2023-01-08 22:09:18] 1 row affected in 6 ms
```

```sql
-- get cinema rooms count
select cinema.name, count(r.id)
from cinema
inner join room r on cinema.id = r.cinema_id
group by cinema.name
order by cinema.name;
```

```sql
select cinema.name, count(r.id)
from cinema
inner join room r on cinema.id = r.cinema_id
group by cinema.name
order by cinema.name;
```

5 rows

	name	count
1	Brekke - Wisoky	3
2	Reichel - Brekke	3
3	Renner and Sons updated	3
4	Schamberger, Heaney and Hyatt	3
5	Torphy - McClure	3

```sql
-- get ticket details
create or replace view ticket_information as
select t.id as "ticket_id",
       t.price as "ticket_price",
       m.title as "movie_title",
       s.row as "seat_row",
       s.number as "seat_number",
       r.name as "room_name",
       to_char(e.starts_at, 'dd.mm.yyyyThh:mm') as "event_starts_at",
       (m.duration / 60.0 / 60)::numeric(3,1) as "movie_duration"
    from ticket as t
inner join event as e on e.id = t.event_id
inner join seat as s on t.seat_id = s.id
inner join movie as m on e.movie_id = m.id
inner join room r on e.room_id = r.id;

select * from ticket_information where ticket_id =
'd76b814e-0807-4c45-a98b-758d1d18e793';
```

1 row | Tx: Auto | DDL | CSV

	ticket_id	tick...	movie_title	sea...	se...	room_name	event_start...	movie_duration
1	d76b814e-0807-4c…	5016	Gislason - Grant	9	9	Osinski Group	27.01.2023T12:01	1.8

```sql
-- cinema location
select "c".name, "a".city, "a".street, "a".house_number
    from "cinema" as "c"
inner join address as "a" on "a".id = c.address_id;
```


```sql
-- cinema location
select "c".name, "a".city, "a".street, "a".house_number
    from "cinema" as "c"
inner join address as "a" on "a".id = c.address_id;
```


5 rows | CSV

	name	city	street	house_number
1	Reichel - Brekke	Ceres	Dietrich Branch	050
2	Brekke - Wisoky	Fort Wayne	Glenna Heights	772
3	Torphy - McClure	Thorastad	Max Pines	0324
4	Schamberger, Heaney and Hyatt	Fort Alvisville	Wyman Harbor	1219
5	Renner and Sons updated	Maple Grove	Josiah Junctions	7292

```sql
-- average bill
select c.id, concat(c.first_name, ' ', c.last_name) as "name",
avg(calculate_order_total(o.id))::int as "avg_order_total"

    from "customer" as "c"
inner join "order" as "o" on "o".customer_id = c.id
group by c.id, "name"
order by "name";
```

```sql
-- average bill
select c.id, concat(c.first_name, ' ', c.last_name) as "name", avg(calculate_order_total( ordr_id: o.id))::i

    from "customer" as "c"
inner join "order" as "o" on "o".customer_id = c.id
group by c.id, "name"
order by "name";
```

99 rows — CSV

	id	name	avg_order_total
1	a7ea1817-f780-480d-bd63-20ba071de69f	Agustin Bode	804241
2	4489e6dd-098a-448d-ac7c-b9cf8d38bca8	Althea Bartoletti	524671
3	7052e3d9-d262-4e1d-9f32-735a1977b6e9	Alysson Price	195313
4	724d97fd-6c46-4700-a6f7-c43be000b185	Andrew Bode	556220
5	5333ae75-7824-4046-8fbf-0e214ca8dfd2	Andy Bernhard	531745
6	2f2145af-c379-4f86-89d1-d21507294ec2	Arielle Kiehn	450889
7	b7bf01af-f7d4-4494-a926-1ed6b0b6382b	Ariel Lowe	545077

```sql
-- event info
select "m".title, "r".name, "c".name, event.starts_at::date
from "event"
         inner join movie m on event.movie_id = m.id
         inner join room r on r.id = event.room_id
         inner join cinema c on r.cinema_id = c.id
where event.id = '2ee806a4-f705-46f1-84de-ee09b97f035f';
```

```sql
-- event info
select "m".title, "r".name, "c".name, event.starts_at::date
from "event"
         inner join movie m on event.movie_id = m.id
         inner join room r on r.id = event.room_id
         inner join cinema c on r.cinema_id = c.id
where event.id = '2ee806a4-f705-46f1-84de-ee09b97f035f';
```

1 row — CSV

	title	name	name	starts_at
1	Gislason - Grant	Osinski Group	Schamberger, Heaney and Hyatt	2023-01-27

6.2 Оптимізації

```
explain analyse
select *
from ticket
inner join event e on ticket.event_id = e.id
inner join movie m on e.movie_id = m.id
where m.id = '3f9903cf-b4f3-45a6-8c21-d90240fad475'
```

13 rows CSV

	QUERY PLAN
7	-> Hash (cost=3.25..3.25 rows=8 width=64) (actual time=0.019..0.020 r…
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on event e (cost=0.00..3.25 rows=8 width=64) (actua…
10	Filter: (movie_id = '3f9903cf-b4f3-45a6-8c21-d90240fad475':…
11	Rows Removed by Filter: 92
12	Planning Time: 0.310 ms
13	Execution Time: 3.584 ms

```
create index idx_ticket_event_id on ticket (event_id);
create index idx_event_movie_id on event (movie_id);

explain analyse
select *
from ticket
inner join event e on ticket.event_id = e.id
inner join movie m on e.movie_id = m.id
where m.id = '3f9903cf-b4f3-45a6-8c21-d90240fad475'
```

13 rows

	QUERY PLAN
7	-> Hash (cost=3.25..3.25 rows=8 width=64)
8	Buckets: 1024 Batches: 1 Memory Usage
9	-> Seq Scan on event e (cost=0.00..3
10	Filter: (movie_id = '3f9903cf-b4
11	Rows Removed by Filter: 92
12	Planning Time: 12.312 ms
13	Execution Time: 1.543 ms

Висновок

Протягом виконання курсової роботи, я створили базу даних для кінотеатру, що є значною оптимізацією процесів. Також, проектування БД – важлива навичка для кожного розробника ПЗ.

Підсумки виконаного завдання:

- Опис предметного середовища
- ER-діаграма
- Реляційну схема
- Структура бази даних
- Функції, тригери, запити
- Запити для будь-яких потреб
- Оптимізації

Перелік посилань

1. Complete SQL and Databases Bootcamp: Zero to Mastery [2023] | Udemy
2. GitHub репозиторій
3. https://www.postgresql.org/
4. https://www.postgresqltutorial.com/
5. https://stackoverflow.com/questions