POLITECHNIKA WROCŁAWSKA

BAZY DANYCH 2

PROJEKT

Prowadzący: Dr inż. Tomasz Babczyński

Temat:

System do zarządzania sprzedażą biletów w multikinie

Kod grupy: K00-52h

Termin zajęć: Poniedziałek 15:15

Nr grupy: 2

Skład grupy: Lena Kuźma, Jan Kucharski

ITE, W4N

12 grudnia, 2022

Spis treści

Opis zasobów ludzkich i obecnych procesów

Celem zaimplementowania systemu jest ułatwienie i zoptymalizowanie dotychczasowego sposobu nabywania biletów kinowych. Program wpłynie na sprawniejsze zarządzanie kinem, pracę obsługi oraz pozytywne doświadczenie klienta.

System umożliwi klientom rezerwację i kupno biletów na wybrane przez nich filmy. Każdy bilet będzie reprezentowany przez następujące dane: cena, sala kinowa, nr fotela, godzina seansu, tytuł filmu, rodzaj biletu (ulgowy, normalny).

Pracownik kasy biletowej będzie mógł potwierdzać rezerwację i sprzedawać bilety klientom. Będzie miał dostęp do informacji o wolnych i zajętych miejscach na poszczególne seanse.

Zarządca kina ma wgląd do bazy danych oraz ustala terminy seansów wraz z salami w których się odbywają. Jest również w stanie ustalić nową cenę biletów.

Dane techniczne

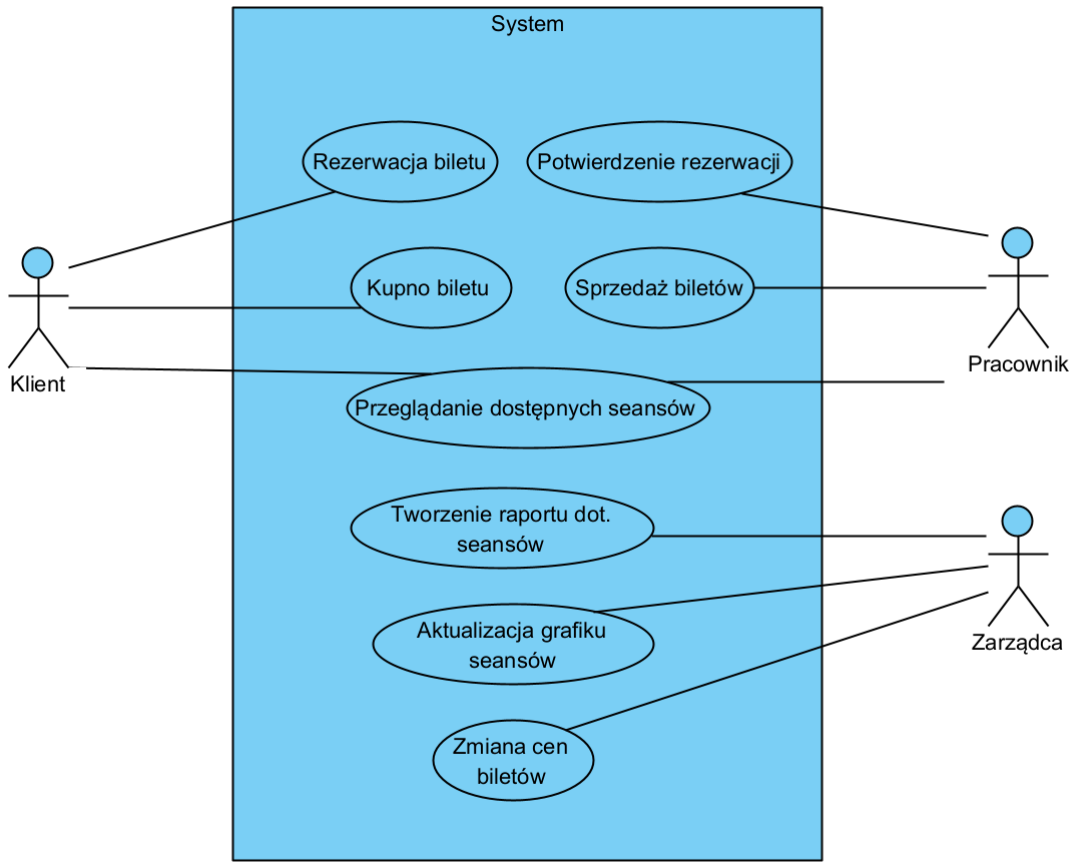
Klient może używać systemu za pośrednictwem aplikacji internetowej, pracownicy kina mają dostęp do wersji firmowej. Multikino składa się z kilku ośrodków w różnych miastach na terenie województwa. W bazie danych przechowujemy informacje o jednostkach kinowych, aktualnie oferowanych filmach, terminach seansów oraz dostępnych, jak i sprzedanych biletach. Informacje te są aktualizowane w każdym tygodniu. Zakładana ilość seansów w ciągu jednego tygodnia to około 300 w jednym kinie (Przy 5 ośrodkach daje to 1500 projekcji). Przy 300 dostępnych miejscach na każdym seansie daje to 450 000 możliwych do sprzedania biletów.

Wymagania funkcjonalne

1. Klient może kupić lub zarezerwować bilet (bez wnoszenia dodatkowej opłaty), ze wskazaniem konkretnego miejsca w sali kinowej, o ile to miejsce jest dostępne. Rezerwacja biletu jest ważna do dnia seansu, aby ją potwierdzić klient musi dokonać opłaty.
2. Pracownik kina może sprzedawać bilety oraz potwierdzać rezerwacje (przyjmując opłatę).
3. Baza aktualnych seansów oraz dostępności miejsc jest widoczna dla wszystkich korzystających z systemu.
4. Zarządca kina ustala grafik seansów na kolejne tygodnie, może wprowadzać aktualizacje z przynajmniej tygodniowym wyprzedzeniem.

Wymagania niefunkcjonalne

1. System musi zapewnić niezawodność w zakresie przechowywania danych oraz bieżących aktualizacji.
2. Aplikacja powinna mieć czytelny i intuicyjny interfejs użytkownika.


System
Rezerwacja biletu
Potwierdzenie rezerwacji
Kupno biletu
Sprzedaż biletów
Przeglądanie dostępnych seansów
Tworzenie raportu dot. seansów
Aktualizacja grafiku seansów
Zmiana cen biletów
Klient
Pracownik
Zarządca

1. **Rezerwacja biletu**

Klient ma możliwość rezerwacji biletu z tygodniowym wyprzedzeniem. W tym celu korzysta z opcji rezerwacji biletu w aplikacji.

Warunki wstępne: W systemie muszą być dostępne wolne miejsca na wybrany przez klienta seans.

Warunki końcowe: Zarezerwowane miejsce nie jest już dostępne dla pozostałych klientów.

2. **Kupno biletu**

Klient ma możliwość kupna biletu. W tym celu korzysta z opcji kupna biletu w aplikacji.

Warunki wstępne: W systemie muszą być dostępne wolne miejsca na wybrany przez klienta seans.

Warunki końcowe: Zakupione miejsce nie jest już dostępne dla pozostałych klientów.

3. **Przeglądanie dostępnych seansów**

Klient lub pracownik mogą przeglądać dostępne terminy i lokalizacje seansów oraz dostępność miejsc. Mają możliwość filtrowania seansów po tytule filmu, lokalizacji i terminie. W tym celu korzystają z opcji przeglądania w aplikacji.

Warunki wstępne: brak

Warunki końcowe: brak

4. **Potwierdzenie rezerwacji**

Pracownik może potwierdzić wcześniej wykonaną przez klienta rezerwację.

Warunki wstępne: W systemie istnieje rezerwacja. Rezerwacja dotyczy seansu odbywającego się przynajmniej 30 min później.

Warunki końcowe: Miejsce podlegające rezerwacji nie jest już dostępne dla pozostałych klientów.

5. **Sprzedaż biletów**

Pracownik ma możliwość sprzedaży biletów po wniesieniu przez klienta opłaty.

Warunki wstępne: W systemie są dostępne miejsca na wybrany seans.

Warunki końcowe: Zakupione miejsce nie jest już dostępne dla pozostałych klientów.

6. Tworzenie raportu dot. seansów

Zarządca ma dostęp do informacji o terminach i lokalizacjach seansów, aktualnego repertuaru, ilości zarezerwowanych miejsc.

Warunki wstępne: Pomyślne zalogowanie do systemu.

Warunki końcowe: Wygenerowanie raportu.

7. Aktualizacja grafiku seansów

Zarządca ma możliwość usuwania i dodawania seansów do repertuaru poszczególnych kin na kolejny tydzień.

Warunki wstępne: Pomyślne zalogowanie do systemu.

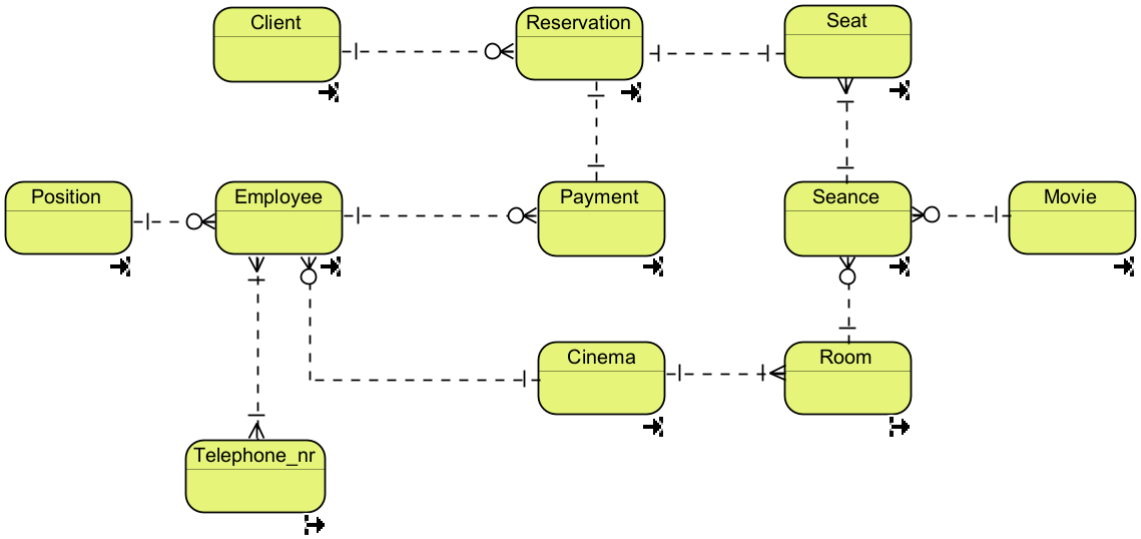
Warunki końcowe: Zaktualizowany repertuar.

8. Zmiana cen biletów

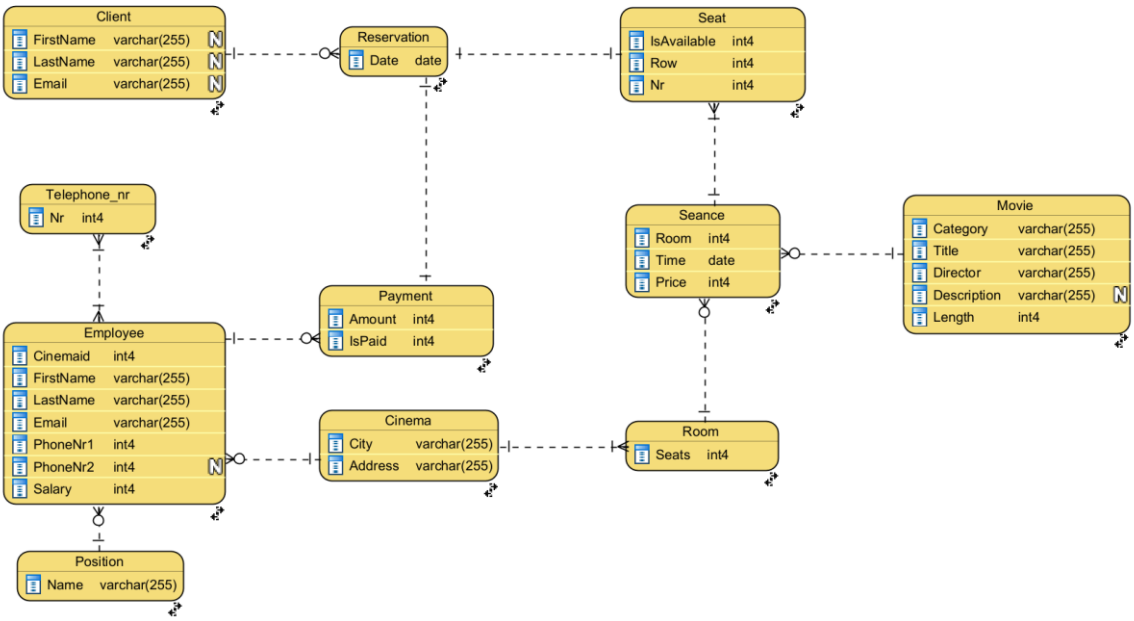
Zarządca ma możliwość aktualizowania cen biletów.

Warunki wstępne: Pomyślne zalogowanie do systemu.

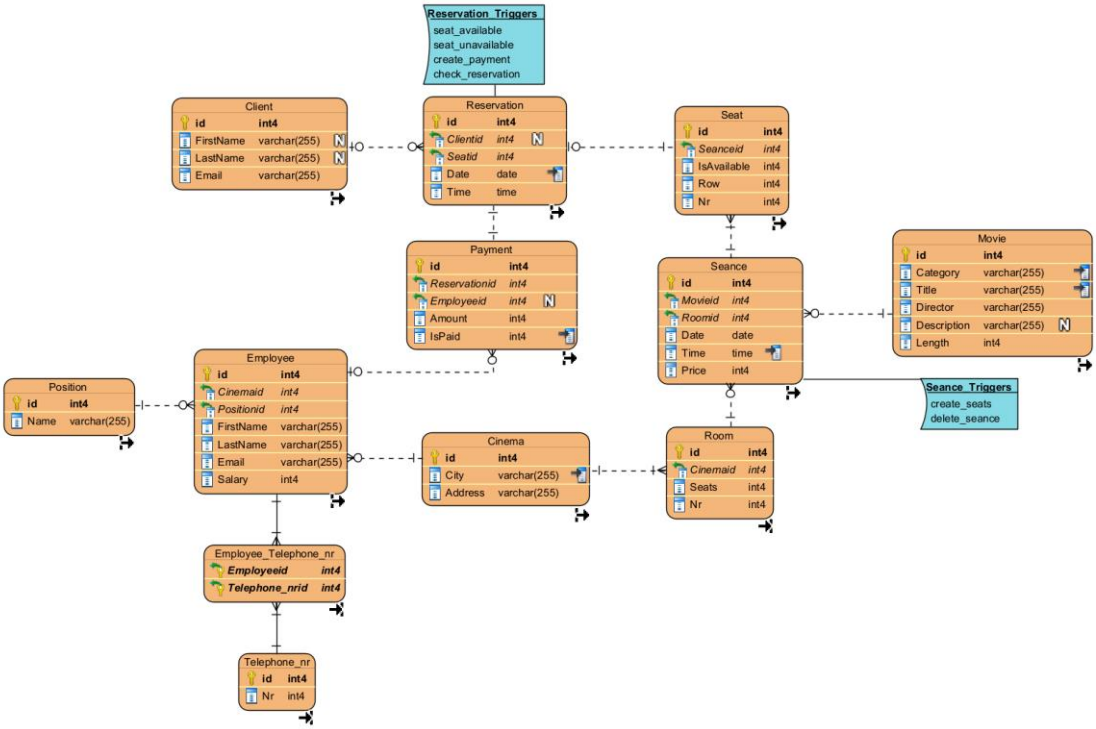
Warunki końcowe: Podwyżka lub obniżka cen biletów.



Rysunek 1: Diagram konceptualny (ERD)



Rysunek 2: Diagram logiczny (ERD)



Rysunek 3: Diagram fizyczny (ERD)

Analiza wielkości tabel

Nazwa tabeli	Ilość rekordów
Client	~ 450 000
Reservation	~ 450 000
Payment	~ 450 000
Employee	~ 150
Position	~ 2
Seat	~ 450 000
Seance	~ 1 500
Movie	~ 20
Cinema	~ 5
Room	~ 10
Telephone_nr	~ 200
Employee_Telephone_nr	~ 250

Tabela 1: Estymowana wielkość tabel w bazie danych

Przy rezerwacji i kupnie biletu system najczęściej będzie odwoływał się do danych (usuwał i dodawał rekordy) z tabel: *Client*, *Reservation, Payment* oraz *Seat.*

Ze względu na sprawdzanie ważności rezerwacji (będzie odbywało się codziennie) często będzie sprawdzana kolumna *Date* w tabeli *Reservation* oraz kolumna *IsPaid* w tabeli *Payment.* Skutkiem wyzwalacza może być również usunięcie rekordów z tabeli *Reservation* i *Payment* oraz modyfikacja kolumny I*sAvailable* w tabeli *Seat*.

Podczas wyszukiwania dostępnych seansów przez klienta często będziemy wyszukiwać kolumny: *City* z tabeli *Cinema, Title* i *Category* z tabeli *Movie* oraz *Time* z tabeli *Seance*.

Największa zmienność przechowywanych danych będzie występowała w tabelach:

- *Reservation* (częste dodawanie rezerwacji)
- *Payment* (częste dodawanie płatności)
- *Client* (częste dodawanie nowych klientów)

Podsumowanie:

Ze względu na dużą ilość danych oraz częste odwoływanie się do tabeli *Seat* potrzebne będzie zastosowanie dla niej indeksów. Indeksy dla pozostałych tabel o podobnej wielkości (*Client, Reservation, Payment*) nie zostaną wprowadzone, ponieważ na danych w nich przechowywanych często będą dokonywane operacje dodawania i usuwania.

Ze względu na częste odwoływanie się do wybranych kolumn (*Tabela 2*), zostaną zastosowane dla nich indeksy.

Nazwa kolumny	Nazwa tabeli
Date	*Reservation*
IsPaid	*Payment*
City	*Cinema*
Title	*Movie*
Category	*Movie*
Time	*Seance*

Tabela 2: Kolumny, dla których zostaną zastosowane indeksy

Potrzebna klauzula typu *cascade-delete*:

- Usuwamy *Seance* → w tabeli *Seat* usuwamy siedzenia przypisane do seansu.
- Usuwamy *Movie* → w tabeli *Seance* usuwamy seansy przypisane do filmu.
- Usuwamy *Seat* → w tabeli *Reservation* usuwamy rezerwację przypisaną do siedzenia.
- Usuwamy *Reservation* → w tabeli *Payment* usuwamy płatność przypisaną do rezerwacji.
- Usuwamy *Employee* → usuwamy rekord w tabeli *Employee_Telephone_nr.*
- Usuwamy *Telephone_nr* → usuwamy rekord w tabeli *Employee_Telephone_nr.*

Potrzebne wyzwalacze:

- *seat_available* – Wyzwalany przy usuwaniu rezerwacji. Zmienia zawartość kolumny *isAvailable* w tabeli *Seat* (0 → 1).
- seat_unavailable – Wyzwalany przy dodawaniu rezerwacji. Zmienia zawartość kolumny *isAvailable* w tabeli *Seat* (1 → 0).
- *create_payment* – Wyzwalany po stworzeniu rezerwacji. Tworzy rekord w tabeli *Payment* przechowujący między innymi informację czy rezerwacja została opłacona.
- *create_seats* – Wyzwalany podczas dodania seansu. Tworzy rekordy w tabeli *Seat* reprezentujące dostępne miejsca na seans.
- *delete_seance* – Wyzwalany po zakończeniu trwania seansu. Usuwa seans z bazy danych.
- *check_reservation* – Wyzwalany 30 minut przed rozpoczęciem planowanego seansu. Jeśli rezerwacja nie została opłacona zostaje usunięta.

Analiza bezpieczeństwa i poufności

Potrzebne role:

- Pracownik obsługujący sprzedaż (Employee)
- Zarządca (Manager)

	Poziom uprawnień	
Nazwa tabeli	Pracownik obsługujący sprzedaż	Zarządca
Payment	Odczyt / Modyfikacja / Dodawanie	Brak
Reservation	Odczyt / Dodawanie	Brak
Seance	Odczyt	Odczyt / Dodawanie / Usuwanie / Modyfikacja
Movie	Odczyt	Odczyt/ Dodawanie / Usuwanie / Modyfikacja
Seat	Odczyt / Modyfikacja	Odczyt
Cinema	Brak	Brak
Employee	Brak	Odczyt /Dodawanie / Usuwanie / Modyfikacja
Position	Brak	Odczyt / Dodawanie / Usuwanie / Modyfikacja
Client	Odczyt	Odczyt
Room	Brak	Brak
Telephone_nr	Brak	Odczyt / Dodawanie / Usuwanie
Employee_Telephone_nr	Brak	Odczyt

Rysunek 4: Poziom uprawnień ról potrzebnych do dostępu do bazy danych

Implementacja bazy danych

Baza danych została wygenerowana na podstawie fizycznego diagramu ERD (*Rysunek3*) w programie *Visual Paradigm*.

Do pracy na bazie danych został wykorzystany *pgAdmin* (program open source do zarządzania i pracy użytkownika z bazą danych *PostgreSQL*).

Do wygenerowania danych testowych wykorzystano internetowy generator: Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel.

1. Sprawdzenie poprawności klauzuli *cascade-delete* (*Reservation → Payment)* oraz wyzwalacza *seat_available* (Usunięcie rezerwacji):

   a. Przykładowy rekord w tabeli *Reservation*, na którym zostaną przeprowadzone testy:

id [PK] integer	clientid integer	seatid integer	Date date	time time without time zone
7	648	178016	2021-12-17	02:40:00

   b. Przypisana do rezerwacji (id = 7) płatność przed usunięciem rezerwacji:

```
SELECT * FROM Payment WHERE Reservationid = 7;
```

id [PK] integer	reservationid integer	employeeid integer	amount integer	ispaid integer
5	7	[null]	33	0

   c. Przypisane do rezerwacji (id = 7) miejsce przed usunięciem rezerwacji:

```
SELECT * FROM Seat WHERE id = 178016;
```

id [PK] integer	seanceid integer	isavailable integer	row integer	nr integer
178016	717	0	1	22

   d. Usunięcie rezerwacji z id =7:

```
DELETE FROM Reservation WHERE id = 7;
```

```
NOTICE:  seat_available sie wykonuje...
DELETE 1

Query returned successfully in 73 msec.
```

   e. Przypisana do rezerwacji (id = 7) płatność po usunięciu rezerwacji nie istnieje:

```
SELECT * FROM Payment WHERE Reservationid = 7;
```

   Zapytanie nie zwróciło żadnego rekordu. Klauzula *cascade-delete* zadziałała poprawnie.

f. Przypisane do rezerwacji (id = 7) miejsce po usunięciu rezerwacji:

```
SELECT * FROM Seat WHERE id = 178016;
```

id [PK] integer	seanceid integer	isavailable integer	row integer	nr integer
178016	717	1	1	22

Zawartość kolumny *isAvailable* zmieniła wartość. Wyzwalacz *seat_available* zadziałał poprawnie.

2. Sprawdzenie poprawności wyzwalaczy *create_payment* oraz *seat_unavailable* (Dodanie rezerwacji, *PU: Rezerwacja biletu*):

a. Dodanie nowego rekordu do tabeli *Reservation*:

```
INSERT INTO reservation(clientid, seatid, "Date")
VALUES (1, 200000, now());
```

```
NOTICE:  create_payment() sie wykonuje...
NOTICE:  seat_unavailable() sie wykonuje...
INSERT 0 1

Query returned successfully in 88 msec.
```

b. Wyświetlenie dodanego rekordu:

```
SELECT * FROM Reservation WHERE Clientid = 1;
```

id [PK] integer	clientid integer	seatid integer	Date date	time time without time zone
1003	1	200000	2022-12-12	[null]

c. Dla nowej rezerwacji (id = 1003) został stworzony rekord w tabeli *Payment*:

```
SELECT * FROM Payment WHERE Reservationid = 1003;
```

id [PK] integer	reservationid integer	employeeid integer	amount integer	ispaid integer
1001	1003	[null]	20	0

Wyzwalacz *create_payment* zadziałał poprawnie.

d. Rekord w tabeli *Seat* (id = 1003) do którego została przypisana rezerwacja:

```
SELECT * FROM Seat WHERE id = 200000;
```

id [PK] integer	seanceid integer	isavailable integer	row integer	nr integer
200000	807	0	1	20

Zawartość kolumny *isAvailable* zmieniła się z 1 na 0. Wyzwalacz *seat_unavailable* zadziałał poprawnie.

3. Pozostałe zapytania, które pozwoliły sprawdzić poprawność implementacji bazy danych:

a. Dodanie nowego seansu (*PU: Aktualizacja grafiku seansów*):

```
INSERT INTO seance(movieid, roomid, "Date", "time", price)
VALUES (1, 1, TO_DATE('2023/09/27', 'yyyy/mm/dd'), '12:00:00', 30);
```

```
NOTICE:  create_seats() sie wykonuje...
INSERT 0 1

Query returned successfully in 78 msec.
```

b. Wyszukanie dostępności miejsc na wybrany film (*PU: Przeglądanie dostępnych seansów*):

```
SELECT m.title, seat.nr, seat.row, seat.isavailable FROM seance as s
INNER JOIN seat ON seat.seanceid = s.id
INNER JOIN movie AS m ON m.title = 'Rooster Cogburn';
```

Pierwsze trzy zwrócone rekordy:

title character varying (255)	nr integer	row integer	isavailable integer
Rooster Cogburn	1	1	1
Rooster Cogburn	2	1	1
Rooster Cogburn	3	1	1

c. Zaktualizowanie statusu płatności *(PU: Potwierdzenie rezerwacji)*:

Rekord w tabeli *Payment* przed aktualizacją:

id [PK] integer	reservationid integer	employeeid integer	amount integer	ispaid integer
6	8	[null]	21	0

```
UPDATE payment
SET ispaid = 1
WHERE reservationid = 8;
```

Po aktualizacji:

id [PK] integer	reservationid integer	employeeid integer	amount integer	ispaid integer
6	8	[null]	21	1

d. Sprawdzenie liczby zarezerwowanych miejsc na dany seans *(PU: Wygeneruj raport)*:

```
SELECT COUNT(id)
FROM seat
WHERE isavailable = 0 AND seanceid = 8;
```

count bigint
3

e. Aktualizacja ceny biletów na dany seans *(PU: Zmiana cen biletów)*:

Rekord w tabeli *Seance* przed aktualizacją:

id [PK] integer	movieid integer	roomid integer	time time without time zone	price integer	Date date
5	12	29	09:47:00	29	2022-02-03

```
UPDATE seance
SET price = 35
WHERE id = 5;
```

Po aktualizacji:

id [PK] integer	movieid integer	roomid integer	time time without time zone	price integer	Date date
5	12	29	09:47:00	35	2022-02-03

Poziom uprawnień: *Manager*

```
SELECT * FROM payment;
```

```
ERROR:  permission denied for table payment
SQL state: 42501
```

Manager nie ma dostępu do tabeli *Payment*. Wynik poprawny.

```
DELETE FROM seance WHERE id = 45;
```

```
DELETE 1

Query returned successfully in 55 msec.
```

Manager może modyfikować tabelę *Seance*. Wynik poprawny.

```
UPDATE seat SET isavailable = 0 WHERE id = 1;
```

```
ERROR:  permission denied for table seat
SQL state: 42501
```

Manager nie może modyfikować tabeli *Seat*. Wynik poprawny.

```
SELECT * FROM seat;
```

id [PK] integer	seanceid integer	isavailable integer	row integer	nr integer
1	2	1	1	1
2	2	1	1	2
3	2	1	1	3

Manager ma dostęp do odczytu tabeli *Seat*. Wynik poprawny.

```
INSERT INTO telephone_nr (nr) VALUES (234567890);
```

```
INSERT 0 1

Query returned successfully in 44 msec.
```

Manager może dodawać i usuwać rekordy z tabeli *Telephone_nr*. Wynik poprawny.

Poziom uprawnień: *Employee*

```
UPDATE payment SET ispaid = 1 WHERE id = 7;
```

```
UPDATE 1

Query returned successfully in 59 msec.
```

Pracownik (*Employee*) może aktualizować rekordy tabeli *Payment*. Wynik poprawny.

```
INSERT INTO reservation (clientid, seatid, "Date", "time")
VALUES (1, 1, CURRENT_DATE, CURRENT_TIME);
```

```
NOTICE:  create_payment() sie wykonuje...
NOTICE:  seat_unavailable() sie wykonuje...
INSERT 0 1

Query returned successfully in 76 msec.
```

Pracownik może dodawać rekordy do tabeli *Reservation*. Wynik poprawny.

```
SELECT * FROM client;
```

id [PK] integer	firstname character varying (255)	lastname character varying (255)	email character varying (255)
1	Demetre	Shawley	dshawley0@shop-pro.jp
2	Peg	Routh	prouth1@cbc.ca
3	Ellie	Glossop	eglossop2@mac.com

Pracownik ma wgląd do tabeli *Client*. Wynik poprawny.

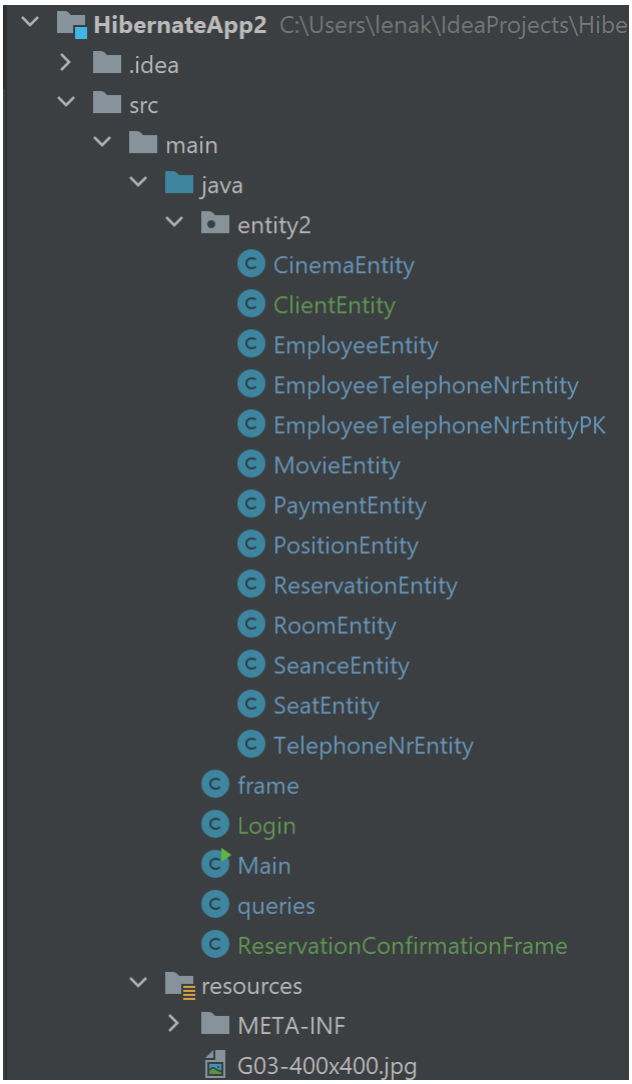
```
SELECT * FROM employee;
```

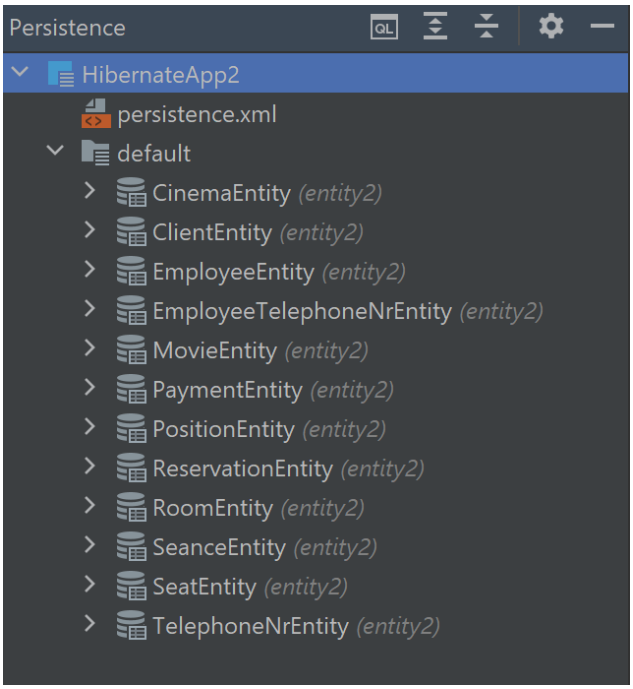
```
ERROR:  permission denied for table employee
SQL state: 42501
```

Pracownik nie ma dostępu do tabeli *Employee*. Wynik poprawny.

Do stworzenia aplikacji wykorzystana została *Java*, a do translacji danych z relacyjnej bazy *Hibernate*. Przy użyciu technologii *Swing* stworzony został interfejs dla pracownika, umożliwiający sprzedaż biletów.

Struktura aplikacji

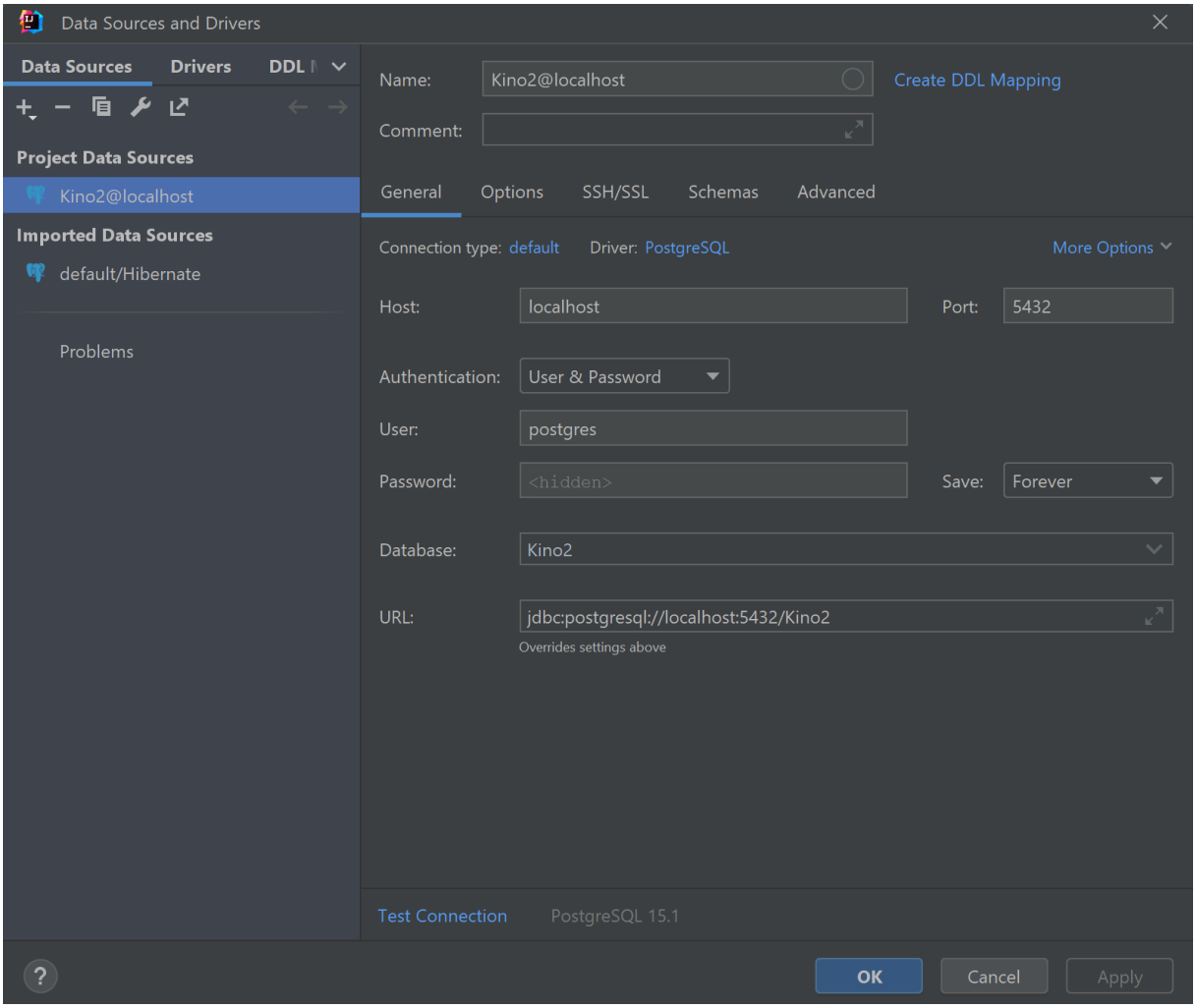




Link do repozytorium

https://github.com/Jaskarnet/HibernateApp2.git

Połączenie z bazą danych



Przykładowa metoda realizująca dostęp do bazy danych

```
1 usage
public static String findMovie(EntityManager entityManager, int movieId){
    Query browseSeances = entityManager.createNativeQuery( s: "SELECT title FROM movie WHERE id=:movieId");
    browseSeances.setParameter( s: "movieId", movieId);
    return browseSeances.getSingleResult().toString();
}
```

Zrzuty ekranu przedstawiające działanie aplikacji