

Bazy Danych 2

Projekt Baza danych kina



1. Wstęp

1.1 Cel projektu

Celem projektu jest stworzenie aplikacji bazodanowej dla małego kina składającego się z trzech sal kinowych. W kinie zatrudnionych jest kilku pracowników oraz jeden menedżer.

1.2 Zakres projektu

Projekt z Baz Danych 2 obejmuje w swoim zakresie:

- utworzenie projektu i struktury bazy danych, wraz z mechanizmami zapewniającymi poprawność przechowywanych informacji, oraz kontroli dostępu do danych,
- implementację zaprojektowanej bazy danych w wybranym środowisku zarządzania bazą danych (DBSM) oraz przeprowadzeniu na niej odpowiednich testów,
- implementację aplikacji bazodanowej w wybranym środowisku programistycznym i przeprowadzenie na niej odpowiednich testów.

2. Analiza wymagań

2.1 Opis słowny systemu

Baza danych, stworzona na potrzeby projektu, powstanie dla niewielkiego kina znajdującego się w niewielkiej kilkutysięcznej miejscowości. Zatrudnionych jest w nim kilkunastu pracowników, pracujących na różnych stanowiskach (m. in. kontrola i sprzedaż biletów, sprzątanie po seansach). Kino zarządzane jest przez menadżera, zajmującego się tworzeniem repertuaru na najbliższe tygodnie oraz zarządzaniem personelem. Tworzona aplikacja, ma usprawnić działanie kina, poprzez umożliwienie klientom dokonywania wcześniejszych rezerwacji, zautomatyzowanie procesu sprzedaży biletów i pomoc w zarządzaniu kinem menadżerowi. W tworzonej aplikacji, nie uwzględniono obecnych restrykcji kinowych, powstałych z powodu trwającej pandemii.

2.2 Wymagania funkcjonalne

Dla różnych użytkowników przypisano konkretne funkcje:

Klient (gość) :

- przeglądanie seansów,
- możliwość utworzenia konta na stronie kina.

Klient (zalogowany) :

- rezerwacja miejsc na seansach i możliwość ich odwołania,
- te same funkcje co Klient(gość).

Pracownik :

- sprzedaż biletów (kasa biletowa),
- odwoływanie rezerwacji dla wszystkich klientów,
- wprowadzenie rachunku (automatycznie podczas sprzedaży),
- te same funkcje co Klient.

Menadżer :

- tworzy/modyfikuje/odwołuje seanse,
- dodaje/modyfikuje/usuwa pracownika,
- dodaj/modyfikuj/usuń film,
- przeglądanie rachunków,
- te same funkcje co pracownik.

2.3 Wymagania niefunkcjonalne

2.3.1 Wykorzystane technologie i narzędzia

- Aplikacja klienta będzie uruchamiana z poziomu Windowsa,
- aplikacja webowa napisana w języku Java hostowana na zewnętrznym serwerze,
- baza danych MySQL,
- modelowanie baz danych, jak i inne potrzebne diagramy UML utworzone zostaną w Visual Paradigm Community Edition,
- program do tworzenia makiet: Figma

2.3.2 Wymagania dotyczące rozmiaru bazy danych

- **Użytkownik:** około kilka tys.
- **Pracownik:** kilku do kilkunastu.
- **Film:** kilka do kilkunastu (skala tygodni).
- **Seanse:** 45-70 tygodniowo.
- **Sale:** 3.
- **Bilety:** max. 840-1000 dziennie, średnio ok. 500-600 dziennie.
- **Rezerwacje:** max. 5880-7000 tygodniowo, średnio ok. 1750-2100 tygodniowo.
- **Loginy:** kilka tysięcy.
- **Rachunki:** jeśli ma przechowywać dane wszystkich zakupów kina w historii jego funkcjonowania, może osiągnąć wartość kilku do nawet kilkunastu tysięcy rekordów. Dziennie tyle co biletów (może być mniej, jeśli kupiono więcej biletów naraz).

2.3.3 Wymagania dotyczące bezpieczeństwa systemu

Planowane zabezpieczenia bazy danych to login i hasło. Aplikacja będzie też chroniła przed dublowaniem zawartości rekordów (np. uniemożliwienie dokonania dwóch rezerwacji na to samo miejsce).

2.4 Przjęte założenia projektowe

W aplikacji umożliwiona będzie:

- rezerwacja biletów z poziomu użytkownika,
- obsługa kina przez pracowników,
- zarządzanie kinem z poziomu menadżera.

Informacje o seansach będą dostępne:

- Dla klientów: do tygodnia przed seansem,
- Dla pracowników i menadżera: dwa tygodnie przed seansem (menadżer może tworzyć seanse w tym okresie).

Zakładamy utworzenie następujących encji:

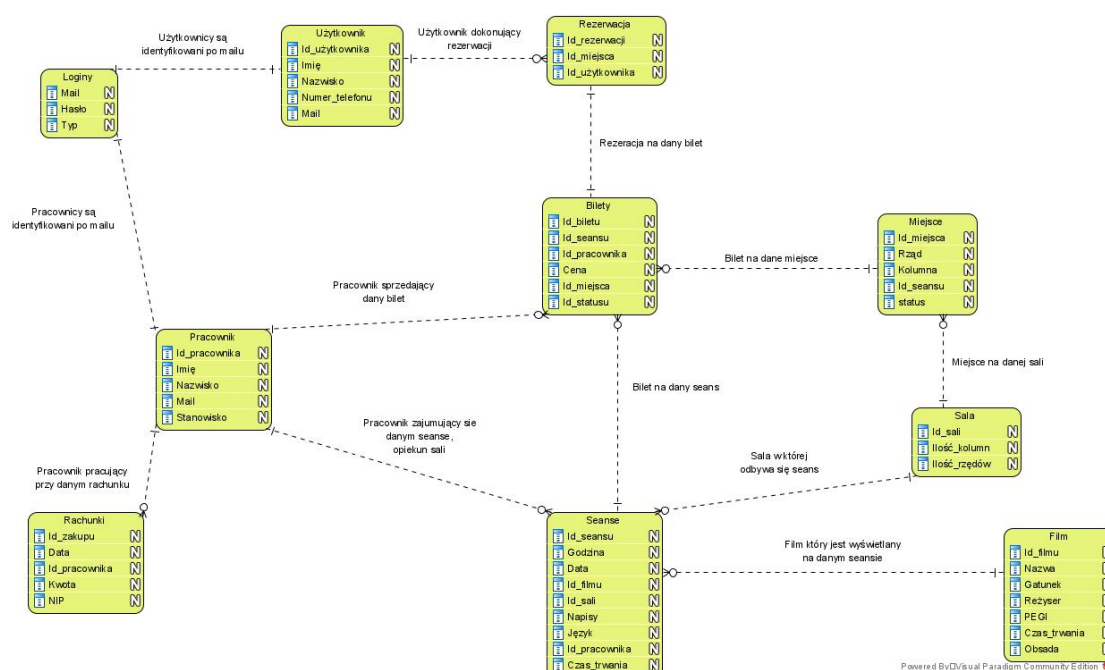
- **Użytkownik** (id_użytkownika, imię, nazwisko, numer telefonu, mail),
- **Pracownik** (id_pracownika, imię, nazwisko, mail, stanowisko pracy),
- **Film** (id_filmu, nazwa, gatunek, reżyser, pegi, czas trwania),
- **Seanse** (id_seansu, godzina, data, id_filmu, id_sali, napisy, język, opiekun, czas trwania),
- **Sala** (id_Sali, ilość rzędów, ilość kolumn),
- **Bilety** (id_biletu, id_seansu, id_pracownika, rząd, miejsce, cena),
- **Rezerwacje** (id_rezerwacji, id_seansu, id_klienta, rząd, miejsce),
- **Loginy** (mail, hasło, typ),
- **Rachunki** (id_zakupu, data, id_pracownika (który dokonał sprzedaży), kwota).

3. Projekt systemu

Projekt i struktury bazy danych, mechanizmów zapewniania poprawności przechowywanych informacji, oraz kontroli dostępu do danych.

3.1 Projekt bazy danych

3.1.1 Analiza rzeczywistości i model konceptualny

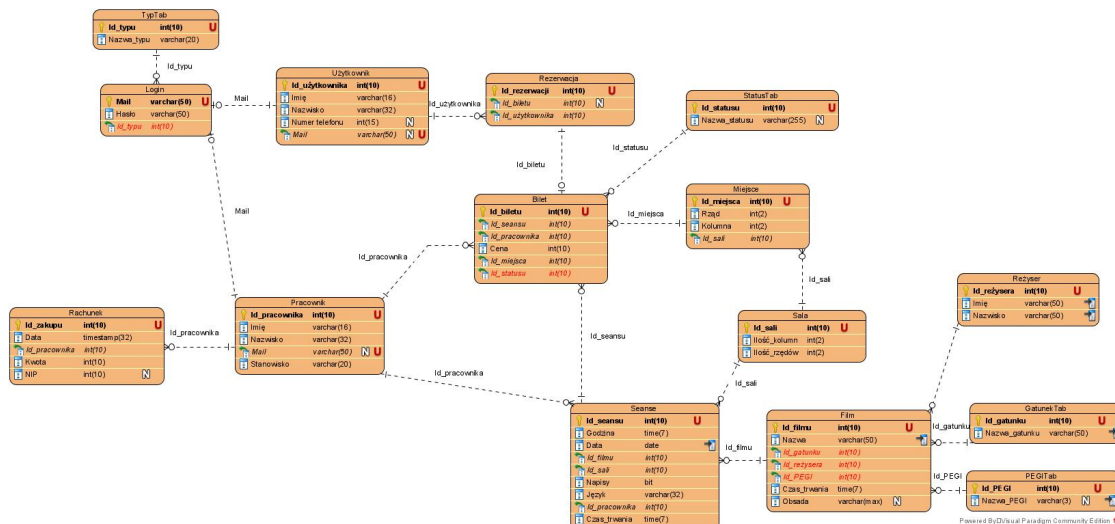


Rysunek 1: Model konceptualny bazy danych.

3.1.2 Model logiczny i normalizacja

Model logiczny i fizyczny są w naszym projekcie tożsame.

3.1.3 Model fizyczny i ograniczenia integralności danych



Rysunek 2: Model fizyczny bazy danych.

3.1.4 Inne elementy schematu - mechanizmy przetwarzania danych

3.1.4.1 Widoki

Widok pracowników (dostępny dla menadżera):

z tabeli Pracownik:

- imię,
- nazwisko,
- mail,
- stanowisko.

Widok filmów (dostępny dla każdego typu użytkownika):

z tabeli Film:

- nazwa,
- reżyser,
- czas trwania,
- obsada,

z tabeli słownikowych:

- PEGI,
- gatunek.

Widok seansów dla klientów (dostępny dla klientów):

z tabeli Seanse:

- godzina,
- data,
- id_sali (w praktyce numer sali),
- napisy,
- język,
- czas trwania,

z tabeli Film:

- nazwa.

Widok seansów dla pracowników i menadżera:

z tabeli Seanse:

- godzina,
- data,
- id_sali (w praktyce numer sali),
- napisy,
- język,
- czas trwania,

z tabeli Film:

- nazwa,

z tabeli Pracownik:

- imię,
- nazwisko,
- mail.

Widok rachunków (widoczny dla menadżera):

z tabeli Rachunek:

- id_zakupy,
- data,
- kwota,
- NIP.

Widok rezerwacji:

z tabeli Rezerwacja:

- id_rezerwacji,

z tabeli Seanse:

- data,
- godzina,

z tabeli Film:

- nazwam

z tabeli Miejsce:

- rząd,
- kolumna.

3.1.4.2 Triggery

- przy CREATE w seansie CREATE wszystkie bilety na seans w danej sali o statusie wolnym,
- przy UPDATE biletu na status Zajęty/Sprzedany, CREATE rachunek.

3.1.4.3 Indeksy

Klucze główne jako indeksy, dodatkowo indeksy dla łatwiejszego wyszukiwania po:

- nazwie filmu,
- imieniu i nazwisku reżysera,
- gatunku filmu,
- ograniczeniach wiekowych dla filmu,
- dacie emisji.

3.1.4.4 Sekwencje

Mechanizm sekwencji zostanie użyty do generacji wszystkich numerów id, przy pomocy autoinkrementacji.

3.1.4.5 Procedury składowane

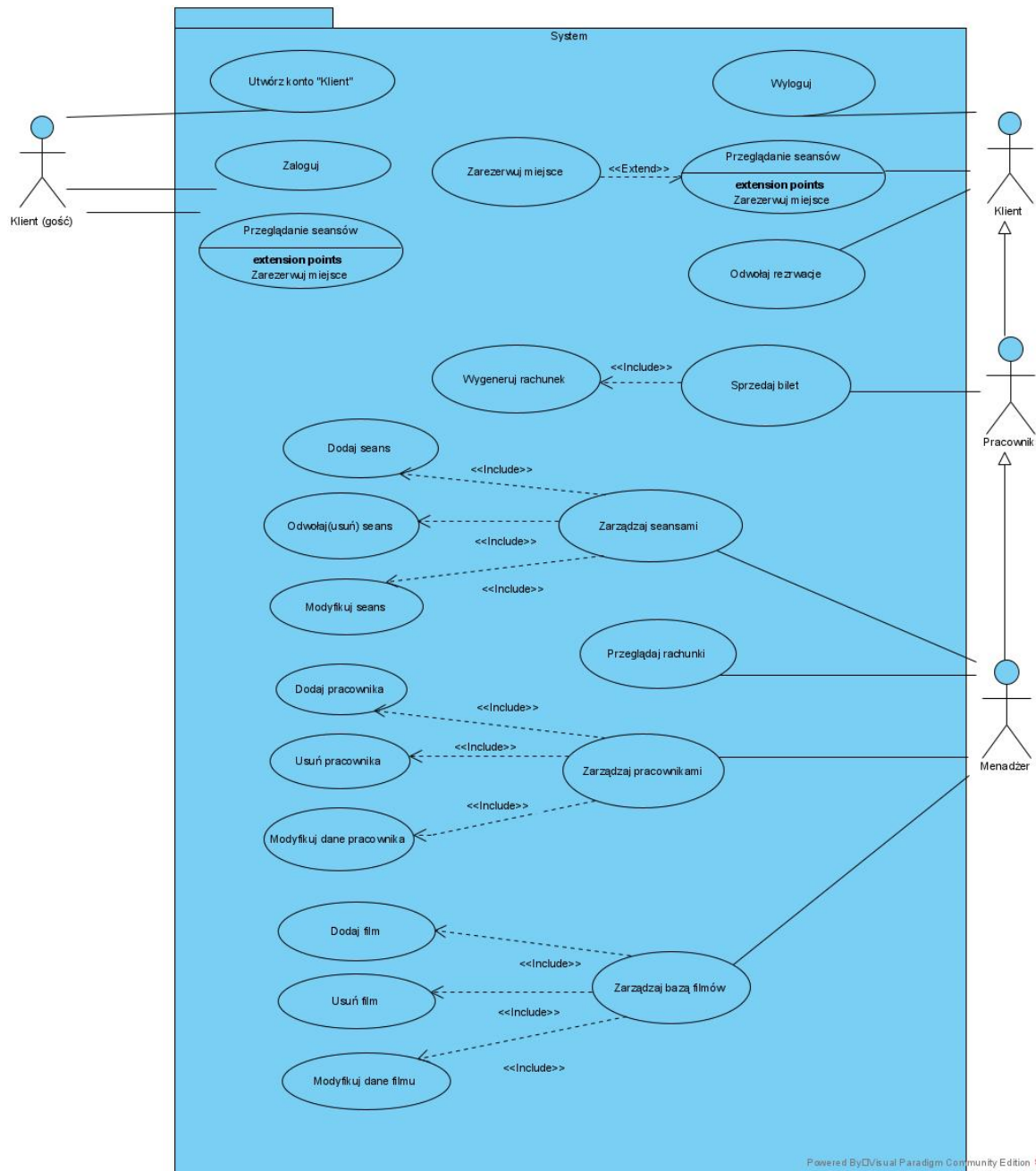
Wykorzystywane przy tabelach słownikowych.

3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

- Do zalogowania się wymagane są login i hasło,
- w zależności od rodzaju logowania (użytkownik, pracownik czy menadżer) wyświetlane są inne panele,
- dla użytkowników widoczne są dane z tabel Seanse, Bilet, Rezerwacja, Film z wyłączeniem danych dotyczących pracowników (pracownik_id),
- użytkownik może edytować dane z tabeli Rezerwacja przypisane do jego konta,
- dla pracowników widoczne są wszystkie dane które są widoczne dla użytkowników plus dodatkowe dane o seansach (dane z tabeli pracownik),
- pracownik może edytować dane z tabeli rezerwacja,
- dla menadżera widoczne i możliwe do edycji jest wszystko.

3.2 Projekt aplikacji użytkownika

3.2.1 Diagram przypadków użycia



Rysunek 3: Diagram przypadków użycia.

3.2.2 Interfejs graficzny i struktura menu

KINO HORYZONTÓW

[Panel użytkownika](#)
Adam Kowalski
123@listopad.pl

[Repertuar](#)
[Cennik](#)
[Kontakt](#)

Moje rezerwacje

Wyloguj

Repertuar

Tytuł	Język	Data	Godzina	Czas trwania	Sala	
lorem ipsum	lorem ipsum	10/10/2020	09:15-09:45am	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/12/2020	12:00-12:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/13/2020	01:15-01:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/14/2020	02:00-02:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/15/2020	12:00-12:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/17/2020	01:15-01:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/17/2020	02:00-02:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/18/2020	09:15-09:45am	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/19/2020	12:00-12:45pm	lorem ipsum	0	zarezerwuj
lorem ipsum	lorem ipsum	10/20/2020	09:15-09:45am	lorem ipsum	0	zarezerwuj

Rysunek 4: Projekt graficzny aplikacji (okno repertuar).

KINO HORYZONTÓW

[Panel menadżera](#)
Adam Kowalski
123@listopad.pl

[Zarządzanie seansami](#)
[Zarządzanie pracownikami](#)
[Zarządzanie filmami](#)
[Zarządzanie rezerwacjami](#)

Przegląd rachunków

[Dodaj seans](#)
[Modyfikuj seans](#)
[Usuń seans](#)

Przełącz konto

Wyloguj

Film

Opiekun

Sala

Data

Godzina

Czas trwania

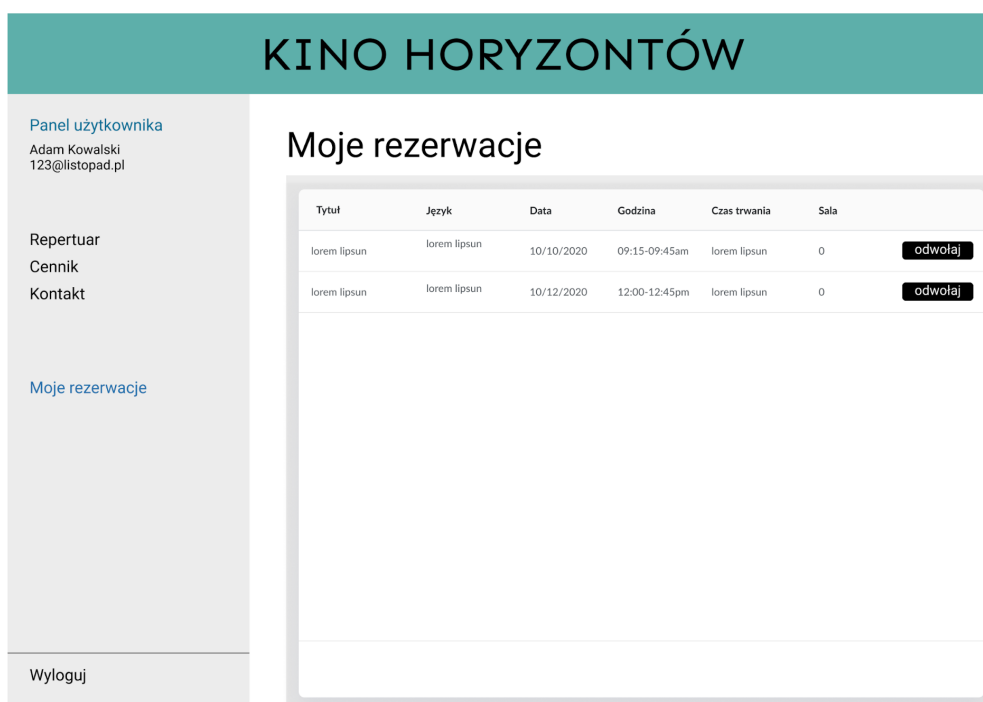
Napisy

Język

[Dodaj](#)

[Anuluj](#)

Rysunek 5: Projekt graficzny aplikacji(okno dodaj seans).



Rysunek 6: Projekt graficzny aplikacji(okno moje rezerwacje).



Rysunek 7: Projekt graficzny aplikacji (okno rejestracja).

3.2.3 Metoda podłączania do bazy danych - integracja z bazą danych

Do integracji aplikacji z bazą danych, planowane jest użycie interfejsu JDBC, które pozwala na komunikowanie się aplikacji napisanych w języku z Java i utworzonej bazy danych, przy pomocy języka SQL. Integracja interfejsu odbywa się w następujących krokach:

1. Zainstaluj lub zlokalizuj bazę danych.
2. Zaimportuj JDBC Into do tworzonej aplikacji.
3. Dodaj sterowniki JDBC do classpath-a aplikacji.
4. Użyj biblioteki JDBC do utworzenia połączenia z bazą danych.
5. Użyj połączenia do wysyłania komend języka SQL.
6. Zakończ połączenie.

4. Implementacja systemu baz danych

4.1 Tworzenie tabel i definiowanie ograniczeń

Listing 1: Deklaracja tabeli *Bilet*.

```
1 CREATE TABLE 'bilet' (  
2   'Id_biletu' int NOT NULL AUTO_INCREMENT,  
3   'Id_seansu' int NOT NULL,  
4   'Id_pracownika' int NOT NULL,  
5   'Cena' int NOT NULL,  
6   'Id_miejsca' int NOT NULL,  
7   'Id_statusu' int NOT NULL,  
8   PRIMARY KEY ('Id_biletu'),  
9   KEY 'Id_miejsca' ('Id_miejsca'),  
10  KEY 'Id_pracownika3' ('Id_pracownika'),  
11  KEY 'Id_seansu' ('Id_seansu'),  
12  KEY 'Id_statusu' ('Id_statusu'),  
13  CONSTRAINT 'Id_miejsca' FOREIGN KEY ('Id_miejsca') REFERENCES '  
    miejsce' ('Id_miejsca'),  
14  CONSTRAINT 'Id_pracownika3' FOREIGN KEY ('Id_pracownika') REFERENCES  
    'pracownik' ('Id_pracownika'),  
15  CONSTRAINT 'Id_seansu' FOREIGN KEY ('Id_seansu') REFERENCES 'seanse'  
    ('Id_seansu'),  
16  CONSTRAINT 'Id_statusu' FOREIGN KEY ('Id_statusu') REFERENCES '  
    statustab' ('Id_statusu'))
```

Listing 2: Deklaracja tabeli *Film*.

```
1 CREATE TABLE 'film' (  
2   'Id_filmu' int NOT NULL AUTO_INCREMENT,  
3   'Nazwa' varchar(50) NOT NULL,  
4   'Id_gatunku' int NOT NULL,  
5   'Id_rezysera' int NOT NULL,  
6   'Id_PEGI' int NOT NULL,  
7   'Czas_trwania' time(6) NOT NULL,  
8   'Obsada' varchar(255) DEFAULT NULL,  
9   PRIMARY KEY ('Id_filmu'),  
10  KEY 'Nazwa' ('Nazwa'),  
11  KEY 'FKFilm519609' ('Id_rezysera'),  
12  KEY 'Id_gatunku' ('Id_gatunku'),  
13  KEY 'Id_PEGI' ('Id_PEGI'),  
14  CONSTRAINT 'FKFilm519609' FOREIGN KEY ('Id_rezysera') REFERENCES '  
    rezyser' ('Id_rezysera'),  
15  CONSTRAINT 'Id_gatunku' FOREIGN KEY ('Id_gatunku') REFERENCES '  
    gatunektab' ('Id_gatunku'),  
16  CONSTRAINT 'Id_PEGI' FOREIGN KEY ('Id_PEGI') REFERENCES 'pegitab' ('  
    Id_PEGI'))
```

Listing 3: Deklaracja tabeli słownikowej *GatunekTab*.

```
1 CREATE TABLE 'gatunektab' (  
2   'Id_gatunku' int NOT NULL AUTO_INCREMENT,  
3   'Nazwa_gatunku' varchar(50) NOT NULL,  
4   PRIMARY KEY ('Id_gatunku'),  
5   KEY 'Nazwa_gatunku' ('Nazwa_gatunku'))
```

Listing 4: Deklaracja tabeli *Login*.

```
1 CREATE TABLE 'login' (  
2   'Mail' varchar(50) NOT NULL,  
3   'Haslo' varchar(50) NOT NULL,  
4   'Id_typu' int NOT NULL,  
5   PRIMARY KEY ('Mail'),  
6   KEY 'Id_typu' ('Id_typu'),  
7   CONSTRAINT 'Id_typu' FOREIGN KEY ('Id_typu') REFERENCES 'typtab' ('  
    Id_typu'))
```

Listing 5: Deklaracja tabeli *Miejsce*.

```
1 CREATE TABLE 'miejsce' (  
2   'Id_miejsca' int NOT NULL AUTO_INCREMENT,  
3   'Rzad' int NOT NULL,  
4   'Kolumna' int NOT NULL,  
5   'Id_sali' int NOT NULL,  
6   PRIMARY KEY ('Id_miejsca'),  
7   KEY 'Id_sali2' ('Id_sali'),  
8   CONSTRAINT 'Id_sali2' FOREIGN KEY ('Id_sali') REFERENCES 'sala' ('  
    Id_sali'))
```

Listing 6: Deklaracja tabeli słownikowej *Pegitab*.

```
1 CREATE TABLE 'pegitab' (  
2   'Id_PEGI' int NOT NULL AUTO_INCREMENT,  
3   'Nazwa_PEGI' varchar(3) DEFAULT NULL,  
4   PRIMARY KEY ('Id_PEGI'),  
5   KEY 'Nazwa_PEGI' ('Nazwa_PEGI'))
```

Listing 7: Deklaracja tabeli *Pracownik*.

```
1 CREATE TABLE 'pracownik' (  
2   'Id_pracownika' int NOT NULL AUTO_INCREMENT,  
3   'Imie' varchar(16) NOT NULL,  
4   'Nazwisko' varchar(32) NOT NULL,  
5   'Mail' varchar(50) DEFAULT NULL,  
6   'Stanowisko' varchar(20) NOT NULL,  
7   PRIMARY KEY ('Id_pracownika'),  
8   UNIQUE KEY 'Mail' ('Mail'),  
9   CONSTRAINT 'Mail2' FOREIGN KEY ('Mail') REFERENCES 'login' ('Mail'))
```

Listing 8: Deklaracja tabeli *Rachunek*.

```
1 CREATE TABLE 'rachunek' (  
2   'Id_zakupu' int NOT NULL AUTO_INCREMENT,  
3   'Data' timestamp(6) NOT NULL,  
4   'Id_pracownika' int NOT NULL,  
5   'Kwota' int NOT NULL,  
6   'NIP' int DEFAULT NULL,  
7   PRIMARY KEY ('Id_zakupu'),  
8   KEY 'Id_pracownika' ('Id_pracownika'),  
9   CONSTRAINT 'Id_pracownika' FOREIGN KEY ('Id_pracownika') REFERENCES '  
   pracownik' ('Id_pracownika'))
```

Listing 9: Deklaracja tabeli *Rezerwacja*.

```
1 CREATE TABLE 'rezerwacja' (  
2   'Id_rezerwacji' int NOT NULL AUTO_INCREMENT,  
3   'Id_biletu' int DEFAULT NULL,  
4   'Id_uzytkownika' int NOT NULL,  
5   PRIMARY KEY ('Id_rezerwacji'),  
6   KEY 'Id_biletu' ('Id_biletu'),  
7   KEY 'Id_uzytkownika' ('Id_uzytkownika'),  
8   CONSTRAINT 'Id_biletu' FOREIGN KEY ('Id_biletu') REFERENCES 'bilet'  
   ('Id_biletu'),  
9   CONSTRAINT 'Id_uzytkownika' FOREIGN KEY ('Id_uzytkownika') REFERENCES  
   'uzytkownik' ('Id_uzytkownika'))
```


Listing 10: Deklaracja tabeli *Rezyser*.

```
1 CREATE TABLE 'rezyser' (  
2   'Id_rezysera' int NOT NULL AUTO_INCREMENT,  
3   'Imie' varchar(50) NOT NULL,  
4   'Nazwisko' varchar(50) NOT NULL,  
5   PRIMARY KEY ('Id_rezysera'),  
6   KEY 'Imie+Nazwisko' ('Imie','Nazwisko'))
```

Listing 11: Deklaracja tabeli *Sala*.

```
1 CREATE TABLE 'sala' (  
2   'Id_sali' int NOT NULL AUTO_INCREMENT,  
3   'Ilo_kolumn' int NOT NULL,  
4   'Ilo_rzedow' int NOT NULL,  
5   PRIMARY KEY ('Id_sali'))
```

Listing 12: Deklaracja tabeli *Seanse*.

```
1 CREATE TABLE 'seanse' (  
2   'Id_seansu' int NOT NULL AUTO_INCREMENT,  
3   'Godzina' time(6) NOT NULL,  
4   'Data' date NOT NULL,  
5   'Id_filmu' int NOT NULL,  
6   'Id_sali' int NOT NULL,  
7   'Napisy' bit(1) NOT NULL,  
8   'Jezyk' varchar(32) NOT NULL,  
9   'Id_pracownika' int NOT NULL COMMENT 'Opiekun sali',  
10  'Czas_trwania' time(6) NOT NULL,  
11  PRIMARY KEY ('Id_seansu'),  
12  KEY 'Id_filmu' ('Id_filmu'),  
13  KEY 'Id_pracownika2' ('Id_pracownika'),  
14  KEY 'Id_sali' ('Id_sali'),  
15  KEY 'Data' ('Data' DESC),  
16  CONSTRAINT 'Id_filmu' FOREIGN KEY ('Id_filmu') REFERENCES 'film' (  
    Id_filmu'),  
17  CONSTRAINT 'Id_pracownika2' FOREIGN KEY ('Id_pracownika') REFERENCES  
    'pracownik' ('Id_pracownika'),  
18  CONSTRAINT 'Id_sali' FOREIGN KEY ('Id_sali') REFERENCES 'sala' (  
    Id_sali'))
```

Listing 13: Deklaracja tabeli słownikowej *StatusTab*.

```
1 CREATE TABLE 'statustab' (  
2   'Id_statusu' int NOT NULL AUTO_INCREMENT,  
3   'Nazwa_statusu' varchar(255) DEFAULT NULL,  
4   PRIMARY KEY ('Id_statusu'))
```

Listing 14: Deklaracja tabeli słownikowej *TypTab*.

```
1 CREATE TABLE 'typtab' (  
2   'Id_typu' int NOT NULL AUTO_INCREMENT,  
3   'Nazwa_typu' varchar(20) NOT NULL,  
4   PRIMARY KEY ('Id_typu'))
```

Listing 15: Deklaracja tabeli *Uzytkownik*.

```
1 CREATE TABLE 'uzytkownik' (  
2   'Id_uzytkownika' int NOT NULL AUTO_INCREMENT,  
3   'Imie' varchar(16) NOT NULL,  
4   'Nazwisko' varchar(32) NOT NULL,  
5   'Numer_telefonu' int DEFAULT NULL,  
6   'Mail' varchar(50) DEFAULT NULL,  
7   PRIMARY KEY ('Id_uzytkownika'),  
8   UNIQUE KEY 'Mail' ('Mail'),  
9   CONSTRAINT 'Mail' FOREIGN KEY ('Mail') REFERENCES 'login' ('Mail'))
```

4.2 Implementacja mechanizmów przetwarzania danych

4.2.1 Widoki

Listing 16: Widok pracowników (dostępny dla menadżera).

```
1 CREATE VIEW 'pracownik_view' AS
2     SELECT
3         'pracownik'. 'Id_pracownika' AS 'Id_pracownika',
4         'pracownik'. 'Imie' AS 'Imie',
5         'pracownik'. 'Nazwisko' AS 'Nazwisko',
6         'pracownik'. 'Mail' AS 'Mail',
7         'pracownik'. 'Stanowisko' AS 'Stanowisko'
8     FROM
9         'pracownik'
```

Listing 17: Widok filmów.

```
1 CREATE VIEW 'film_view' AS
2     SELECT
3         'film'. 'Id_filmu' AS 'Id_filmu',
4         'film'. 'Nazwa' AS 'Nazwa',
5         'rezyser'. 'Imie' AS 'Imie',
6         'rezyser'. 'Nazwisko' AS 'Nazwisko',
7         'film'. 'Czas_trwania' AS 'Czas_trwania',
8         'film'. 'Obsada' AS 'Obsada',
9         'pegitab'. 'Nazwa_PEGI' AS 'Nazwa_PEGI',
10        'gatunektab'. 'Nazwa_gatunku' AS 'Nazwa_gatunku'
11    FROM
12        ((( 'film'
13        JOIN 'rezyser' ON (('film'. 'Id_rezysera' = 'rezyser'. '
14        Id_rezysera'))
15        JOIN 'pegitab' ON (('film'. 'Id_PEGI' = 'pegitab'. 'Id_PEGI'))
16        JOIN 'gatunektab' ON (('film'. 'Id_gatunku' = 'gatunektab'. '
17        Id_gatunku'))
```

Listing 18: Widok seansów (klient).

```
1 CREATE VIEW 'seans_klient_view' AS
2     SELECT
3         'seanse'. 'Id_seansu' AS 'Id_seansu',
4         'seanse'. 'Data' AS 'Data',
5         'seanse'. 'Godzina' AS 'Godzina',
6         'seanse'. 'Id_sali' AS 'Id_sali',
7         'seanse'. 'Napisy' AS 'Napisy',
8         'seanse'. 'Jezyk' AS 'Jezyk',
9         'seanse'. 'Czas_trwania' AS 'Czas_Trwania',
10        'film'. 'Nazwa' AS 'Nazwa'
11    FROM
12        ('seanse'
13        JOIN 'film' ON (('seanse'. 'Id_filmu' = 'film'. 'Id_filmu'))
```

Listing 19: Widok seansów (menadżer).

```

1 CREATE VIEW 'seans_pracownik_view' AS
2     SELECT
3         'seanse'.'.Id_seansu' AS 'Id_seansu',
4         'seanse'.'.Data' AS 'Data',
5         'seanse'.'.Godzina' AS 'Godzina',
6         'seanse'.'.Id_sali' AS 'Id_sali',
7         'seanse'.'.Napisy' AS 'Napisy',
8         'seanse'.'.Jezyk' AS 'Jezyk',
9         'seanse'.'.Czas_trwania' AS 'Czas_Trwania',
10        'film'.'.Nazwa' AS 'Nazwa',
11        'pracownik'.'.Mail' AS 'Mail',
12        'pracownik'.'.Imie' AS 'Imie',
13        'pracownik'.'.Nazwisko' AS 'Nazwisko'
14    FROM
15        (('seanse'
16         JOIN 'film' ON (('seanse'.'.Id_filmu' = 'film'.'.Id_filmu'))))
17        JOIN 'pracownik' ON (('seanse'.'.Id_pracownika' = 'pracownik'.'.
18                             Id_pracownika'))

```

Listing 20: Widok rachunków.

```

1 CREATE VIEW 'rachunek_view' AS
2     SELECT
3         'rachunek'.'.Id_zakupu' AS 'Id_zakupu',
4         'rachunek'.'.Data' AS 'Data',
5         'rachunek'.'.Kwota' AS 'Kwota',
6         'rachunek'.'.NIP' AS 'NIP'
7    FROM
8        'rachunek'

```

Listing 21: Widok rezerwacji.

```

1 CREATE VIEW 'rezerwacje_view' AS
2     SELECT
3         'rezerwacja'.'.Id_rezerwacji' AS 'Id_rezerwacji',
4         'seanse'.'.Data' AS 'Data',
5         'seanse'.'.Godzina' AS 'Godzina',
6         'film'.'.Nazwa' AS 'Nazwa',
7         'miejsce'.'.Rzad' AS 'rzad',
8         'miejsce'.'.Kolumna' AS 'kolumna'
9    FROM
10        (((('rezerwacja'
11         JOIN 'bilet' ON (('rezerwacja'.'.Id_biletu' = 'bilet'.'.Id_biletu'
12                          ')))
13         JOIN 'miejsce' ON (('bilet'.'.Id_miejsca' = 'miejsce'.'.
14                           Id_miejsca'))))
15         JOIN 'seanse' ON (('bilet'.'.Id_seansu' = 'seanse'.'.Id_seansu'))
16         JOIN 'film' ON (('seanse'.'.Id_filmu' = 'film'.'.Id_filmu'))))

```

4.2.2 Triggery

Listing 22: Trigger *seanse after insert*.

```
1 CREATE TRIGGER 'seanse_AFTER_INSERT '  
2 AFTER INSERT ON 'seanse '  
3 FOR EACH ROW  
4 BEGIN  
5     DECLARE k INT;  
6     DECLARE w INT;  
7     DECLARE i INT;  
8     DECLARE rzedy INT;  
9     DECLARE kolumny INT;  
10    DECLARE plusindeks INT;  
11    Set k=1;  
12    Set w =1;  
13    Set i = 1;  
14    SET rzedy = (Select distinct ilo _rzedow from Sala join seanse  
                  using(Id_sali) where Sala.Id_sali = New.Id_sali);  
15    SET kolumny = (Select distinct ilo _kolumn from Sala join seanse  
                   using(Id_sali) where Sala.Id_sali = New.Id_sali);  
16  
17    loop_label: LOOP  
18        IF w> rzedy then  
19            LEAVE loop_label;  
20        END IF;  
21  
22        IF k> kolumny then  
23            SET w = w+1;  
24            SET k = 1;  
25            ITERATE loop_label;  
26        END IF;  
27  
28        if new.Id_sali = 1 then  
29            set plusindeks = 0;  
30        end if;  
31        if new.Id_sali = 2 then  
32            set plusindeks = 150;  
33        end if;  
34        if new.Id_sali = 3 then  
35            set plusindeks = 300;  
36        end if;  
37  
38        INSERT INTO Bilet(Id_seansu,Id_pracownika,Cena,Id_miejsca,  
                          Id_statusu)  
39        Values (NEW.Id_Seansu,NEW.Id_Pracownika,20,plusindeks + i,1);  
40        SET i = i +1;  
41        SET k = k +1;  
42    END LOOP loop_label;  
43    END
```

Listing 23: Trigger *rezerwacje after insert.*

```
1 CREATE TRIGGER 'rezerwacja_AFTER_INSERT '  
2 AFTER INSERT ON 'rezerwacja '  
3 FOR EACH ROW  
4 BEGIN  
5     UPDATE Bilet SET Id_statusu=2  
6     WHERE new.id_biletu=bilet.id_biletu;  
7 END
```

Listing 24: Trigger *rezerwacje after delete.*

```
1 CREATE TRIGGER 'rezerwacja_AFTER_DELETE '  
2 AFTER DELETE ON 'rezerwacja '  
3 FOR EACH ROW  
4 BEGIN  
5     UPDATE Bilet SET Id_statusu=1  
6     WHERE old.id_biletu=bilet.id_biletu;  
7 END
```

Listing 25: Trigger *sala after insert.*

```
1 CREATE TRIGGER 'sala_AFTER_INSERT '  
2 AFTER INSERT ON 'sala '  
3 FOR EACH ROW  
4 BEGIN  
5     DECLARE k INT;  
6     DECLARE w INT;  
7     DECLARE rzedy INT;  
8     DECLARE kolumny INT;  
9     Set k=1;  
10    Set w =1;  
11    SET rzedy = new.ilo _rzedow;  
12    SET kolumny = new.ilo _kolumn;  
13  
14    loop_label: LOOP  
15        IF w> rzedy then  
16            LEAVE loop_label;  
17        END IF;  
18  
19        IF k> kolumny then  
20            SET w = w+1;  
21            SET k = 1;  
22            ITERATE loop_label;  
23        END IF;  
24  
25  
26        INSERT INTO miejsce(Rzad,Kolumna,Id_sali)  
27        Values (w,k,new.Id_sali);  
28        SET k = k + 1;  
29    END LOOP loop_label;  
30 END
```

Listing 26: Trigger *bilet update after*.

```

1 CREATE TRIGGER 'bilet_AFTER_UPDATE'
2 AFTER UPDATE ON 'bilet'
3 FOR EACH ROW
4 BEGIN
5     IF New.Id_statusu=3 THEN
6         INSERT INTO rachunek(Data,Id_pracownika,Kwota)
7         VALUES(now(),New.Id_pracownika,New.Cena);
8     END IF;
9 END

```

4.3 Implementacja uprawnień i innych zabezpieczeń

Listing 27: Komendy typu grant.

```

1 CREATE USER 'klient(anonimowy)'@'localhost' IDENTIFIED by 'klient(
   anonimowy)';
2 CREATE USER 'klient'@'localhost' IDENTIFIED by 'klient';
3 CREATE USER 'manager'@'localhost' IDENTIFIED by 'admin';
4 CREATE USER 'pracownik'@'localhost' IDENTIFIED by 'pracownik';
5
6 GRANT SELECT ON bdkino.film_view TO 'klient(anonimowy)'@'localhost';
7 GRANT SELECT ON bdkino.seans_klient_view TO 'klient(anonimowy)'@'
   localhost';
8 GRANT INSERT ON bdkino.uzytownicy TO 'klient(anonimowy)'@'localhost';
9
10 GRANT SELECT ON bdkino.film_view TO 'klient(zalogowany)'@'localhost';
11 GRANT SELECT ON bdkino.seans_klient_view TO 'klient(zalogowany)'@'
   localhost';
12 GRANT SELECT ON bdkino.rezerwacje_view TO 'klient(zalogowany)'@'
   localhost';
13 GRANT INSERT ON bdkino.rezerwacje TO 'klient(zalogowany)'@'localhost';
14 GRANT DELETE ON bdkino.rezerwacje TO 'klient(zalogowany)'@'localhost';
15
16 GRANT SELECT ON bdkino.seans_pracownik_view TO 'pracownik'@'localhost';
17 GRANT SELECT ON bdkino.rezerwacje_view TO 'pracownik'@'localhost';
18 GRANT SELECT ON bdkino.film_view TO 'pracownik'@'localhost';
19 GRANT SELECT ON bdkino.bilety TO 'pracownik'@'localhost';
20 GRANT UPDATE ON bdkino.bilet TO 'pracownik'@'localhost';
21 GRANT UPDATE ON bdkino.rezerwacja TO 'pracownik'@'localhost';
22 GRANT DELETE ON bdkino.rezerwacja TO 'pracownik'@'localhost';
23 GRANT INSERT ON bdkino.rezerwacje TO 'pracownik'@'localhost';
24
25 GRANT ALL PRIVILEGES ON baza.* TO 'manager'@'localhost';

```

4.4 Testowanie bazy danych na przykładowych danych

Listing 28: Tabela słownikowa *PEGITab*.

```
1 insert into PEGITab ( Nazwa_PEGI)
2 VALUES
3 ("3+"), ("7+"), ("12+"), ("16+"), ("18+");
```

	Id_PEGI	Nazwa_PEGI
▶	1	3+
	2	7+
	3	12+
	4	16+
	5	18+
✱	NULL	NULL

Rysunek 8: Rezultat po wpisaniu komend.

Listing 29: Tabela słownikowa *GatunekTab*.

```
1 insert into GatunekTab( Nazwa_gatunku)
2 VALUES
3 ("Komedia"),
4 ("Dramat"),
5 ("Thriller"),
6 ("Sci-Fi"),
7 ("Dokumentalny"),
8 ("Akcji"),
9 ("Romans");
```

	Id_gatunku	Nazwa_gatunku
▶	1	Komedia
	2	Dramat
	3	Thriller
	4	Sci-Fi
	5	Dokumentalny
	6	Akcji
	7	Romans
✱	NULL	NULL

Rysunek 9: Rezultat po wpisaniu komend.

Listing 30: Tabela *Reżyser*.

```

1 insert into Rezyser (Imie, Nazwisko)
2 VALUES
3 ("Quentin", "Tarantino"),
4 ("Stanley", "Kubrick"),
5 ("Chritopher", "Nolan"),
6 ("Martin", "Scorsese"),
7 ("Clint", "Eastwood"),
8 ("Mel", "Gibson"),
9 ("Peter", "Jackson"),
10 ("Larry", "Charles");

```

	Id_rezysera	Imie	Nazwisko
▶	1	Quentin	Tarantino
	2	Stanley	Kubrick
	3	Chritopher	Nolan
	4	Martin	Scorsese
	5	Clint	Eastwood
	6	Mel	Gibson
	7	Peter	Jackson
	8	Larry	Charles
*	NULL	NULL	NULL

Rysunek 10: Rezultat po wpisaniu komend.

Listing 31: Tabela słownikowa *StatusTab*.

```

1 insert into StatusTab (Nazwa_statusu)
2 VALUES
3 ("Wolne"),
4 ("Zarezerwowane"),
5 ("Zajete");

```

	Id_statusu	Nazwa_statusu
▶	1	Wolne
	2	Zarezerwowane
	3	Zajete
*	NULL	NULL

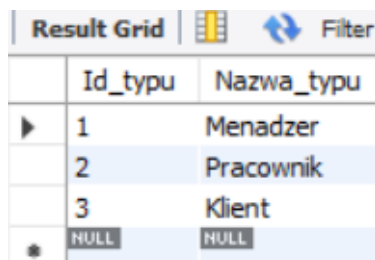
Rysunek 11: Rezultat po wpisaniu komend.

Listing 32: Tabela słownikowa *typTab*.

```

1 INSERT   typtab(Nazwa_typu) VALUES ("Menadzer");
2 INSERT   typtab(Nazwa_typu) VALUES ("Pracownik");
3 INSERT   typtab(Nazwa_typu) VALUES ("Klient");

```



	Id_typu	Nazwa_typu
▶	1	Menadzer
	2	Pracownik
	3	Klient
★	NULL	NULL

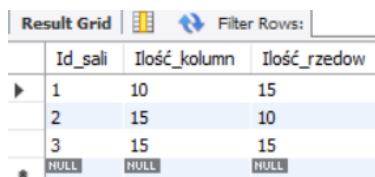
Rysunek 12: Rezultat po wpisaniu komend.

Listing 33: Tabela *Sala*.

```

1 insert into Sala(ilo _kolumn , ilo _rzedow)
2 VALUES
3 (10,15),
4 (15,15),
5 (15,10);

```



	Id_sali	Ilość_kolumn	Ilość_rzedow
▶	1	10	15
	2	15	10
	3	15	15
★	NULL	NULL	NULL

Rysunek 13: Rezultat po wpisaniu komend.

Listing 34: Tabela *login*.

```

1 insert into login (Mail, Haslo, Id_typu) values ('bazy@danych.pwr', '
  dX5CTl', 2);
2 insert into login (Mail, Haslo, Id_typu) values ('niedziela@grudzien.
  edu', 'BkNFfrE', 2);
3 insert into login (Mail, Haslo, Id_typu) values ('poniedziale@grudzien.
  edu', '9ykNqwx3p3W', 2);
4 insert into login (Mail, Haslo, Id_typu) values ('sroda@grudzien.edu',
  'oshPx7Awf3x', 2);
5 insert into login (Mail, Haslo, Id_typu) values ('czwartek@grudzien.edu
  ', 'uX4R7Q2Mp2yH', 2);
6 insert into login (Mail, Haslo, Id_typu) values ('piatek@grudzien.edu',
  '2pdgkmdS', 3);
7 insert into login (Mail, Haslo, Id_typu) values ('poniedzialek@listopad
  .edu', '2pdgkmdS', 1);
8 insert into login (Mail, Haslo, Id_typu) values ('wtorek@listopad.edu',
  'w1TDgU54wt', 1);
9 insert into login (Mail, Haslo, Id_typu) values ('sroda@listopad.edu',
  'wodGzrug', 1);
10 insert into login (Mail, Haslo, Id_typu) values ('czwartek@listopad.edu
  ', 'a8waqn1bZA4', 1);
11 insert into login (Mail, Haslo, Id_typu) values ('piatek@listopad.edu',
  'CBSzq3t1yEwv', 1);
12 insert into login (Mail, Haslo, Id_typu) values ('sobota@listopad.edu',
  'TJLoURKDA4vj', 1);
13 insert into login (Mail, Haslo, Id_typu) values ('niedziela@listopad.
  edu', 'cTMVFXvHNNH', 1);

```

Mail	Haslo	Id_typu
bazy@danych.pwr	dX5CTl	2
czwartek@grudzien.edu	uX4R7Q2Mp2yH	2
czwartek@listopad.edu	a8waqn1bZA4	2
niedziela@grudzien.edu	BkNFfrE	2
niedziela@listopad.edu	cTMVFXvHNNH	1
piatek@grudzien.edu	2pdgkmdS	3
piatek@listopad.edu	CBSzq3t1yEwv	2
poniedziale@grudzien.edu	9ykNqwx3p3W	2
poniedzialek@listopad.edu	2pdgkmdS	3
sobota@listopad.edu	TJLoURKDA4vj	2
sroda@grudzien.edu	oshPx7Awf3x	2
sroda@listopad.edu	wodGzrug	2
wtorek@listopad.edu	w1TDgU54wt	2

Rysunek 14: Rezultat po wpisaniu komend.

Listing 35: Tabela *pracownik*.

```

1 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Bazdy
  ', 'Danych', 'bazy@danych.pwr', 'Kierownik');
2 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Bazdy
  ', 'Danych', 'niedziela@grudzien.edu', 'Pracownik');
3 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Bazdy
  ', 'Danych', 'poniedziale@grudzien.edu', 'Pracownik');
4 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Bazdy
  ', 'Danych', 'sroda@grudzien.edu', 'Pracownik');
5 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Bazdy
  ', 'Danych', 'czwartek@grudzien.edu', 'Manager');
6 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Bazdy
  ', 'Danych', 'piatek@grudzien.edu', 'Kierownik');

```

	Id_pracownika	Imie	Nazwisko	Mail	Stanowisko
1		Bazdy	Danych	bazy@danych.pwr	Kierownik
2		Jan	Dany	niedziela@grudzien.edu	Pracownik
3		Baz	Astral	poniedziale@grudzien.edu	Sprzątac
4		Mikey	Mouse	sroda@grudzien.edu	Pracownik
5		Donald	Duck	czwartek@grudzien.edu	Manager
6		Goofey	Pies	piątek@grudzien.edu	Kierownik
	NULL	NULL	NULL	NULL	NULL

Rysunek 15: Rezultat po wpisaniu komend.

Listing 36: Wprowadzenie błędnych danych do tabeli *pracownik*.

```

1 insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Test'
  , 'Bledu', 'niematakiegomaila@test.pl', 'Kierownik');

```

#	Time	Action	Message	Duration / Fetch
672	21:55:26	insert into pracownik (Imie, Nazwisko, Mail, Stanowisko) values ('Test', 'Bledu', 'niematakiegomaila@test.pl', 'Kierownik')	Error Code: 1452. Cannot add or update a child row: a foreign key constraint f...	0.016 sec

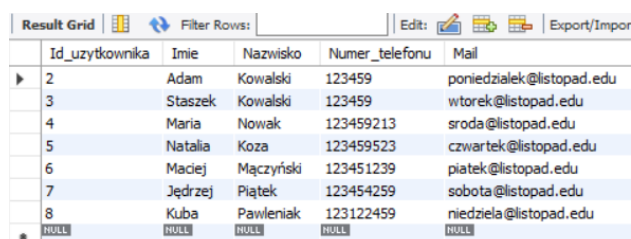
Rysunek 16: Rezultat po wpisaniu komend z błędem.

Listing 37: Tabela *uzytkownik*.

```

1 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Adam', 'Kowalski', 123459, 'poniedzialek@listopad.edu');
2 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Staszek', 'Kowalski', 123459, 'wtorek@listopad.edu');
3 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Maria', 'Nowak', 123459213, 'sroda@listopad.edu');
4 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Natalia', 'Koza', 123459523, 'czwartek@listopad.edu');
5 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Maciej', 'M czy ski', 123451239, 'piatek@listopad.edu');
6 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    J drzej', 'Pi tek', 123454259, 'sobota@listopad.edu');
7 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Kuba', 'Pawleniak', 123122459, 'niedziela@listopad.edu');

```



	Id_uzytkownika	Imie	Nazwisko	Numer_telefonu	Mail
2		Adam	Kowalski	123459	poniedzialek@listopad.edu
3		Staszek	Kowalski	123459	wtorek@listopad.edu
4		Maria	Nowak	123459213	sroda@listopad.edu
5		Natalia	Koza	123459523	czwartek@listopad.edu
6		Maciej	Maczyński	123451239	piatek@listopad.edu
7		Jędrzej	Piątek	123454259	sobota@listopad.edu
8		Kuba	Pawleniak	123122459	niedziela@listopad.edu
*	NULL	NULL	NULL	NULL	NULL

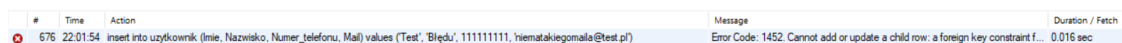
Rysunek 17: Rezultat po wpisaniu komend.

Listing 38: Wprowadzenie błędnych danych do tabeli *uzytkownik*.

```

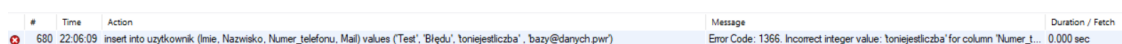
1 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Test', 'Bledu', 111111111, 'niematakiegomaila@test.pl');
2
3 insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('
    Test', 'Bledu', 'toniejestliczba', 'bazy@danych.pwr');

```



#	Time	Action	Message	Duration / Fetch
676	22:01:54	insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('Test', 'Bledu', 111111111, niematakiegomaila@test.pl)	Error Code: 1452. Cannot add or update a child row: a foreign key constraint f...	0.016 sec

Rysunek 18: Rezultat po wpisaniu komendy z błędnym mailem.



#	Time	Action	Message	Duration / Fetch
680	22:06:09	insert into uzytkownik (Imie, Nazwisko, Numer_telefonu, Mail) values ('Test', 'Bledu', 'toniejestliczba', bazy@danych.pwr)	Error Code: 1366. Incorrect integer value: 'toniejestliczba' for column 'Numer_t...	0.000 sec

Rysunek 19: Rezultat po wpisaniu komendy z błędnym numerem telefonu.

Listing 39: Tabela *Filmy*

```

1 insert into film (Nazwa,Id_gatunku, Id_rezysera, Id_PEGI, Czas_trwania,
   Obsada)
2 VALUES
3 ("Borat",5,8,4,'02:00',"ktos na pewno tu gra"),
4 ("Gwiezdne wojny",2,5,3,'02:30:00',"ukasz Niebochodzca"),
5 ("Wladca pierścieni",3,7,3,'03:00:00',"Arazgon"),
6 ("Hobbit",3,7,2,'03:00:00',"Gondolf"),
7 ("Avangers",1,2,1,'03:20:00',"cz owiek elazko "),
8 ("Czerwony Kapturek",7,1,5,'01:10:00',"wilk i babcia");

```

Nazwa	Imie	Nazwisko	Czas_trwania	Obsada	Nazwa_PEGI	Nazwa_gatunku
Gwiezdne wojny	Clint	Eastwood	02:30:00.000000	Łukasz Niebochodzca	12+	Dramat
Władca pierścieni	Peter	Jackson	03:00:00.000000	Arazgon	12+	Thriller
Borat	Larry	Charles	02:00:00.000000	ktos na pewno tu gra	16+	Dokumentalny
Czerwony Kapturek	Quentin	Tarantino	01:10:00.000000	wilk i babcia	18+	Romans
Avangers	Stanley	Kubrick	03:20:00.000000	człowiek żelazko	3+	Komedia
Hobbit	Peter	Jackson	03:00:00.000000	Gondolf	7+	Thriller

Rysunek 20: Rezultat po wpisaniu komend.

Listing 40: Wprowadzenie błędnych danych do tabeli *filmy* (nieistniejące IdPegi).

```

1 insert into film (Nazwa,Id_gatunku, Id_rezysera, Id_PEGI, Czas_trwania,
   Obsada)
2 VALUES
3 ("Avangers",1,2,6,'03:20:00',"cz owiek elazko ");

```

5221 22:23:12 insert into film (Nazwa,Id_gatunku, Id_rezysera, Id_PEGI, Czas_trwania, Obsada) VALUES ("Avangers",1,2,6,"03:20:00","człowiek żelazko ... Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('bdkino`.`film', CONSTRAINT 'Id_PEGI' FOREIGN KE... 0.016 sec

Rysunek 21: Rezultat po wpisaniu błędnej komendy.

Podobnie będzie jeżeli będziemy próbować dane nie odpowiadające niczemu z tabel Idgatunku i Idreżysera.

Listing 41: Tabela *Seanse*

```

1 insert into seanse (Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,
   Czas_trwania,Id_sali)
2 VALUES
3 ('11:20:00','2020-12-13',1,0,"Polski",3,'02:15:00',1),
4 ('17:00:00','2020-12-13',2,1,"Rosyjski",5,'02:45:00',2),
5 ('12:00:00','2020-12-14',3,1,"Angielski",3,'03:15:00',2),
6 ('13:00:00','2020-12-14',4,0,"Polski",2,'03:15:00',2),
7 ('19:30:00','2020-12-14',6,1,"Angielski",4,'01:15:00',3),
8 ('13:00:00','2020-12-16',3,1,"Polski",1,'03:15:00',1);

```

Data	Godzina	Id_sali	Napisy	Jezyk	Czas_Trwania	Nazwa	Mail	Imie	Nazwisko
2020-12-13	11:20:00.000000	1	0	Polski	02:15:00.000000	Borat	poniedziale@grudzien.edu	Baz	Astral
2020-12-13	17:00:00.000000	2	1	Rosyjski	02:45:00.000000	Gwiezdne wojny	czwartek@grudzien.edu	Donald	Duck
2020-12-14	12:00:00.000000	2	1	Angielski	03:15:00.000000	Władca pierścieni	poniedziale@grudzien.edu	Baz	Astral
2020-12-14	13:00:00.000000	2	0	Polski	03:15:00.000000	Hobbit	niedziela@grudzien.edu	Jan	Dany
2020-12-14	19:30:00.000000	3	1	Angielski	01:15:00.000000	Czerwony Kapturek	sroda@grudzien.edu	Mikey	Mouse
2020-12-16	13:00:00.000000	1	1	Polski	03:15:00.000000	Władca pierścieni	bazy@danych.pwr	Bazdy	Danych

Rysunek 22: Rezultat po wpisaniu komend.

Listing 42: Wprowadzenie błędnych danych do tabeli *seanse* (kolejno złe Idfilmu)

```
1 insert into seanse(Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,
   Czas_trwania,Id_sali)
2 VALUES
3 ('11:20:00','2020-12-13',9,0,"Polski",3,'02:15:00',1);
4
5 insert into seanse(Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,
   Czas_trwania,Id_sali)
6 VALUES
7 ('11:20:00','2020-12-13',1,0,"Polski",3,'02:15:00',4);
8
9 insert into seanse(Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,
   Czas_trwania,Id_sali)
10 VALUES
11 ('11:20:00','2020-12-13',1,0,"Polski",35,'02:15:00',1);
```

```
5224 22:29:42 insert into seanse(Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,Czas_trwania,Id_sali) VALUES (11:20:00,'2020-12-13',9,0,"Polski",3,'02:15:00',1); Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('bdkino`.`seanse`, CONSTRAINT 'Id_filmu' FOREIGN K... 0.000 sec
5225 22:29:46 insert into seanse(Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,Czas_trwania,Id_sali) VALUES (11:20:00,'2020-12-13',1,0,"Polski",3,'02:15:00',4); Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('bdkino`.`seanse`, CONSTRAINT 'Id_sali' FOREIGN K... 0.015 sec
5226 22:29:48 insert into seanse(Godzina,Data,Id_filmu,Napisy,Jezyk,Id_pracownika,Czas_trwania,Id_sali) VALUES (11:20:00,'2020-12-13',1,0,"Polski",35,'02:15:00',1); Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('bdkino`.`seanse`, CONSTRAINT 'Id_pracownika2' F... 0.000 sec
```

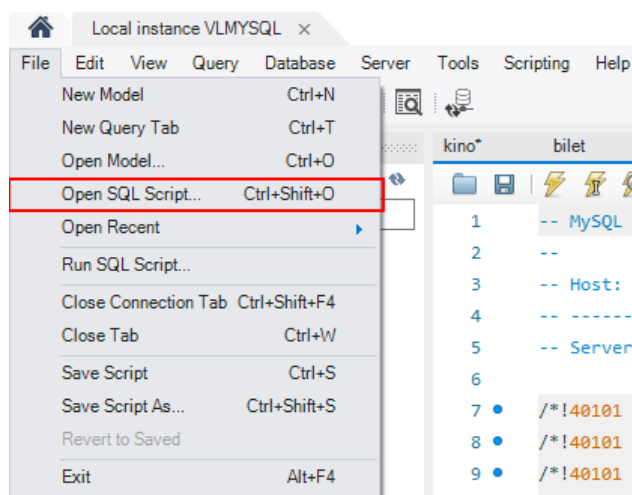
Rysunek 23: Rezultat po wpisaniu błędnych komend.

Testy wydajnościowe przeprowadzono dla maksymalnych wielkości danych podanych w punkcie 2.3.2. Baza działała sprawnie i poprawnie zarówno w kwestii wprowadzania danych jak i ich przeszukiwania.

5. Implementacja i testy aplikacji

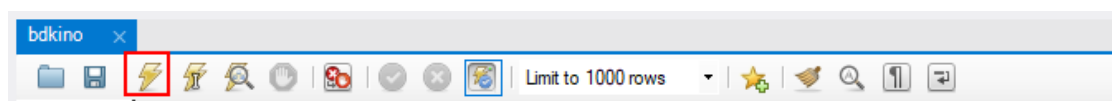
5.1 Instalacja i konfiguracja systemu

By uruchomić aplikację webową, należy wpierw załadować utworzoną w poprzednim punkcie aplikację bazodanową bdkino.sql. W oknie MySQL Workbench, należy załadować plik sql, klikając w menu File, Open Sql Script.



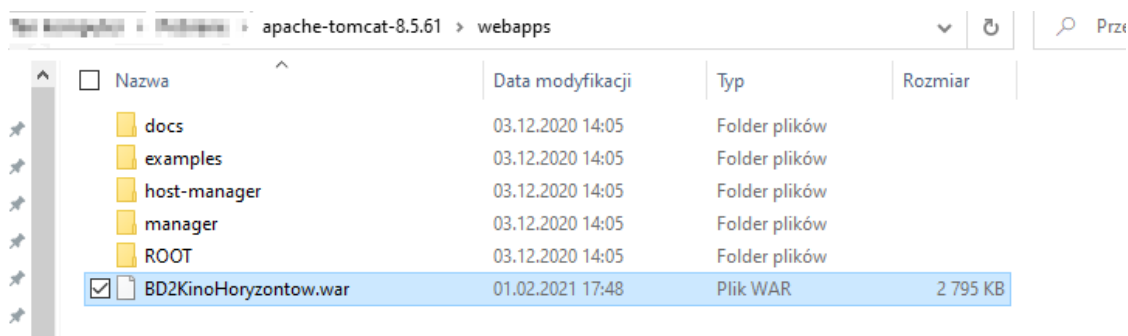
Rysunek 24: Lokalizacja opcji wczytywania skryptu SQL.

Po załadowaniu skryptu do programu, w nowoutworzonym oknie należy kliknąć okienko błyskawicy, by załadować skrypy bazodanowy na serwer MYSQL.



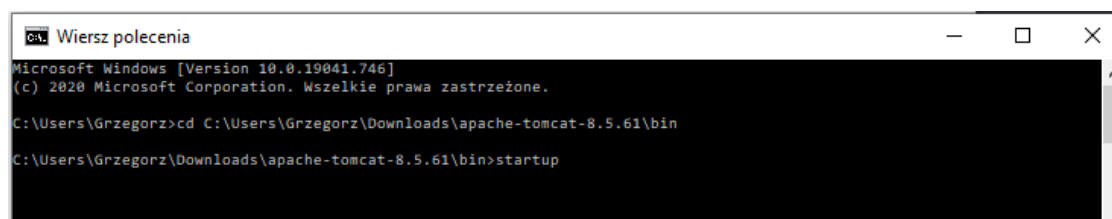
Rysunek 25: Lokalizacja przycisku Execute.

Następnie należy pobrać wybrany serwer, na którym zostanie odpalony aplikacja, znajdująca się pod plikiem .war. Poniżej przedstawiono instrukcji konfiguracji aplikacji na serwerze Tomcat 8.5. Plik *BD2KinoHoryzontow.war*, należy wrzucić do katalogu webapps, katalogu, w którym znajduje się serwer Tomcat.



Rysunek 26: Lokalizacja katalogu.

W kolejnym kroku, przy pomocy Wiersza Poleceń, dostępnego w Windowsie, poprzez wpisanie w wyszukiwarce Menu Start cmd, należy wpisać uruchomić plik startup.bat, analogicznie do przykładu podanego poniżej.



Rysunek 27: Uruchomienie serwera z wykorzystaniem wiersza poleceń.

Po uruchomieniu serwera, by uzyskać dostęp do aplikacji, należy w oknie dowolnej przeglądarki, wpisać frazę: *http://localhost:8080/BD2KinoHoryzontow/*.

5.2 Instrukcja użytkowania aplikacji

5.2.1 Ogólne Poruszanie się po aplikacji

Poruszanie się pomiędzy stronami w aplikacji odbywa się przy użyciu panelu bocznego. Informacje z danej strony wyświetlane są w jej głównej środkowej części. W tym samym miejscu znajdują się także wszystkie formularze które będzie można wypełniać.

5.2.2 Opis możliwości dla klienta niezalogowanego

Dla klienta niezalogowanego dostępne są opcje przeglądania aktualnych seansów, zobaczenia stron zawierających informacje o kontakcie i cenniku. Klient ten ma też opcję zarejestrowania nowego konta, lub zalogowania się na już istniejące.

5.2.3 Opis możliwości dla klienta zalogowanego

Klient zalogowany ma możliwości podobne jak klient niezalogowany (z oczywistym pominięciem możliwości rejestracji i logowania, gdyż zostały już niejako dokonane). Dodatkowo klient zalogowany ma możliwości dokonania rezerwacji na stronie repertuar. Żeby tego dokonać, należy kliknąć przycisk "Zarezerwuj" przy interesującym nas sensie, a następnie wybrać jedno z dostępnych na liście miejsc. Po dokonaniu rezerwacji klient może także zobaczyć ją na liście swoich rezerwacji wraz z innymi, aktualnymi rezerwacjami dokonanyymi przez niego. Będąc w tym miejscu może w dowolnej chwili anulować rezerwację.

5.2.4 Opis możliwości dla pracownika

Możliwości pracownika sprowadzają się do operacji na biletach. Na stronie, na którą przenosi nas przycisk kasa z panelu bocznego, pracownik ma możliwość potwierdzenia rezerwacji (praktycznie sprzedania biletu) na podstawie Id rezerwacji podawanego przez klienta (W teorii sytuacja ta powinna rozgrywać się przy fizycznej kasie w kinie). Poza tym pracownik może także dokonać sprzedaży na wolne miejsce na wybrany seans.

5.2.5 Opis możliwości dla managera

Manager posiada możliwość przeglądania, tworzenia, modyfikowania oraz usuwania filmów, seansów i pracowników. Może także przeglądać utworzone wcześniej rachunki (sprzedane bilety). Aby dokonać zmian należy przejść poprzez panel boczny do konkretnej podstrony, po czym w panelu bocznym wybrać spośród opcji modyfikuj oraz dodaj. Modyfikowanie przenosi nas do strony gdzie przy konkretnym wyniku obecne są 2 przyciski E oraz "D". Pierwszy przenosi nas do formularza modyfikacji (opisane poniżej), drugi usuwa dany wynik. Opcja dodaj prowadzi nas do jednego z formularzy omówionych poniżej.

W kwestii dodawania filmów, należy podać tytuł, reżysera (jeżeli nie istnieje w bazie, zostanie dodany), czas trwania, obsadę, oraz wybrać z dostępnych opcji gatunek i PEGI. Modyfikowanie pozwala na zmianę wszystkich tych wartości.

Dodawanie seansów wymaga wybrania z listy filmu, opiekuna seansu (pracownika z nazwiska), sali, języka w którym film będzie emitowany oraz tego czy napisy będą obecne. Dodatkowo należy podać datę i godzinę seansu, oraz czas jego trwania. Modyfikowanie pozwala na zmianę wszystkich tych wartości poza wybranym filmem oraz salą (zamiast zmieniać te wartości lepiej utworzyć nowy seans, przy zmianie mogłoby się okazać, że klient zarezerwował bilet na inny film niż by chciał, albo w nowej sali zabrakło miejsc).

Zarządzanie pracownikami pozwala dodać pracownika podając jego dane osobowe, dane do logowania oraz stanowisko (jest to pole bardziej opisowe niż funkcyjne). Modyfikowanie pracownika pozwala na modyfikację tych wartości poza email (jest on używany jako identyfikator pracownika).

5.3 Testowanie opracowanych funkcji programistycznych

5.3.1 Wyświetlania pracowników

Kino Horyzontów																																															
Panel menadżera Zarządzanie seansami Zarządzanie pracownikami Zarządzanie filmami Przegląd rachunków Dodaj pracownika Modyfikuj dane Wyloguj	<table> <tr> <th>Imię</th><th>Nazwisko</th><th>Email</th><th>Stanowisko</th></tr> <tr> <td>Bazdy</td><td>Danych</td><td>bazy@danych.pwr</td><td>Kierownik zmiany</td></tr> <tr> <td>Mikey</td><td>Mouse</td><td>sroda@grudzien.edu</td><td>Pracownik</td></tr> <tr> <td>Donald</td><td>Duck</td><td>czwartek@grudzien.edu</td><td>Sprzątac</td></tr> <tr> <td>Goofey</td><td>Pies</td><td>piatek@grudzien.edu</td><td>Kierownik zmiany</td></tr> <tr> <td>Cezary</td><td>Cezaryn</td><td>cezary@cezary.cez</td><td>Kasjer</td></tr> <tr> <td>Janusz</td><td>Bienat</td><td>Janusz@bienat.com</td><td>Sumator</td></tr> <tr> <td>Jonasz</td><td>Koran</td><td>Jonasz@koran.mekka</td><td>Piekarz</td></tr> <tr> <td>Imieprzyklad</td><td>NazwiskoPrzyklad</td><td>imie@nazwisko.com</td><td>operator</td></tr> <tr> <td>JACA</td><td>PRACA</td><td>praca@jaca.work</td><td>praca praca</td></tr> <tr> <td>Pracownik</td><td>PiAkny</td><td>ponury@pracowni.com</td><td>kosiarz</td></tr> </table>			Imię	Nazwisko	Email	Stanowisko	Bazdy	Danych	bazy@danych.pwr	Kierownik zmiany	Mikey	Mouse	sroda@grudzien.edu	Pracownik	Donald	Duck	czwartek@grudzien.edu	Sprzątac	Goofey	Pies	piatek@grudzien.edu	Kierownik zmiany	Cezary	Cezaryn	cezary@cezary.cez	Kasjer	Janusz	Bienat	Janusz@bienat.com	Sumator	Jonasz	Koran	Jonasz@koran.mekka	Piekarz	Imieprzyklad	NazwiskoPrzyklad	imie@nazwisko.com	operator	JACA	PRACA	praca@jaca.work	praca praca	Pracownik	PiAkny	ponury@pracowni.com	kosiarz
Imię	Nazwisko	Email	Stanowisko																																												
Bazdy	Danych	bazy@danych.pwr	Kierownik zmiany																																												
Mikey	Mouse	sroda@grudzien.edu	Pracownik																																												
Donald	Duck	czwartek@grudzien.edu	Sprzątac																																												
Goofey	Pies	piatek@grudzien.edu	Kierownik zmiany																																												
Cezary	Cezaryn	cezary@cezary.cez	Kasjer																																												
Janusz	Bienat	Janusz@bienat.com	Sumator																																												
Jonasz	Koran	Jonasz@koran.mekka	Piekarz																																												
Imieprzyklad	NazwiskoPrzyklad	imie@nazwisko.com	operator																																												
JACA	PRACA	praca@jaca.work	praca praca																																												
Pracownik	PiAkny	ponury@pracowni.com	kosiarz																																												

Rysunek 28: Widok pracowników z poziomu aplikacji

Result Grid				
Filter Rows:				
Id_pracownika	Imie	Nazwisko	Mail	Stanowisko
1	Bazdy	Danych	bazy@danych.pwr	Kierownik zmiany
4	Mikey	Mouse	sroda@grudzien.edu	Pracownik
5	Donald	Duck	czwartek@grudzien.edu	Sprzątac
6	Goofey	Pies	piatek@grudzien.edu	Kierownik zmiany
7	Cezary	Cezaryn	cezary@cezary.cez	Kasjer
8	Janusz	Bienat	Janusz@bienat.com	Sumator
9	Jonasz	Koran	Jonasz@koran.mekka	Piekarz
10	Imieprzyklad	NazwiskoPrzyklad	imie@nazwisko.com	operator
13	JACA	PRACA	praca@jaca.work	praca praca
14	Pracownik	PiAkny	ponury@pracowni.com	kosiarz

Rysunek 29: Widok pracowników z poziomu Bazy danych

Wyniki w obu przypadkach są takie same, czyli program działa tutaj poprawnie.

5.3.2 Wyświetlania filmów

Kino Horyzontów

Panel menadżera

Zarządzanie seansami

Zarządzanie pracownikami

Zarządzanie filmami

Przegląd rachunków

Dodaj film

Modyfikuj dane

Wyloguj

Nazwa	Reżyser	Długość	Gatunek	Pegi	Obsada
Borat	Larry Charles	03:14:15	Komedia	3+	przystojny kazach
Gwiezdne wojny	Clint Eastwood	02:30:00	Dramat	12+	Łukasz Niebochodzka
Władca pierścieni	Peter Jackson	03:00:00	Thriller	12+	Arazgon
Hobbit	Peter Jackson	03:00:00	Thriller	7+	Gondolf
Avangers	Stanley Kubrick	03:20:00	Komedia	3+	człowiek żelazko
Czerwony Kapturek	Quentin Tarantino	01:10:00	Romans	18+	wilk i babcia
W lesie	Mel Gibson	02:21:43	Dramat	3+	Taka jaka była
Nikt a nikt	Clint Eastwood	02:21:43	Dramat	12+	Twoja mama kochana

Rysunek 30: Widok filmów z poziomu aplikacji

Result Grid							
Filter Rows:				Export:			
	Id_filmu	Nazwa	Imie	Nazwisko	Czas_trwania	Obsada	Nazwa_PEGI
1	Borat	Larry	Charles	03:14:15.000000	przystojny kazach	3+	Komedia
2	Gwiezdne wojny	Clint	Eastwood	02:30:00.000000	Łukasz Niebochodzka	12+	Dramat
3	Władca pierścieni	Peter	Jackson	03:00:00.000000	Arazgon	12+	Thriller
4	Hobbit	Peter	Jackson	03:00:00.000000	Gondolf	7+	Thriller
5	Avangers	Stanley	Kubrick	03:20:00.000000	człowiek żelazko	3+	Komedia
6	Czerwony Kapturek	Quentin	Tarantino	01:10:00.000000	wilk i babcia	18+	Romans
7	W lesie	Mel	Gibson	02:21:43.000000	Taka jaka była	3+	Dramat
8	Nikt a nikt	Clint	Eastwood	02:21:43.000000	Twoja mama kochana	12+	Dramat

Rysunek 31: Widok filmów z poziomu Bazy danych

Wyniki w obu przypadkach są takie same, czyli program działa tutaj poprawnie.

5.3.3 Wyświetlania seansów

Kino Horyzontów									
Panel menadżera									
<div>Zarządzanie seansami</div> <div>Zarządzanie pracownikami</div> <div>Zarządzanie filmami</div> <div>Przegląd rachunków</div> <div>Dodaj seans</div> <div>Modyfikuj dane</div> <div>Wyloguj</div>									
Id	Data	Godzina	Film	Sala	Napisy	Język	Długość	Opiekun	
1	2022-06-19	11:25:00	Borat	1	TAK	Polski	03:15:00	Mikey Mouse	
3	2020-12-14	12:00:00	Władca pierścieni	2	TAK	Angielski	03:15:00	Bazdy Danych	
4	2020-12-14	21:00:00	Hobbit	2	TAK	Angielski	03:00:00	Jonasz Koran	
5	2029-12-14	19:30:00	Czerwony Kapturek	3	NIE	Polski	01:30:00	JACA PRACA	
6	2020-12-16	13:00:00	Władca pierścieni	1	TAK	Polski	03:15:00	Bazdy Danych	
7	2021-01-02	14:00:00	Borat	1	NIE	Polski	03:00:00	Bazdy Danych	
8	2222-11-11	13:12:00	Władca pierścieni	2	NIE	Polski	01:30:00	Donald Duck	
12	2025-08-12	16:30:00	Borat	1	NIE	Polski	01:30:00	Bazdy Danych	
13	2029-08-28	19:00:00	Borat	1	TAK	Polski	02:32:00	Bazdy Danych	
14	2028-09-12	15:00:00	Borat	1	NIE	Angielski	03:00:00	Bazdy Danych	

Rysunek 32: Widok seansów dla menadżera z poziomu aplikacji

Result Grid										
Filter Rows:										
Export: Wrap Cell Content: <input checked="" type="checkbox"/>										
Id_seansu	Data	Godzina	Id_sali	Napisy	Język	Czas_Trwania	Nazwa	Mail	Imie	Nazwisko
1	2022-06-19	11:25:00.000000	1	1	Polski	03:15:00.000000	Borat	sroda@grudzien.edu	Mikey	Mouse
3	2020-12-14	12:00:00.000000	2	1	Angielski	03:15:00.000000	Władca pierścieni	bazy@danych.pwr	Bazdy	Danych
4	2020-12-14	21:00:00.000000	2	1	Angielski	03:00:00.000000	Hobbit	Jonasz@koran.mekka	Jonasz	Koran
5	2029-12-14	19:30:00.000000	3	0	Polski	01:30:00.000000	Czerwony Kapturek	praca@jaca.work	JACA	PRACA
6	2020-12-16	13:00:00.000000	1	1	Polski	03:15:00.000000	Władca pierścieni	bazy@danych.pwr	Bazdy	Danych
7	2021-01-02	14:00:00.000000	1	0	Polski	03:00:00.000000	Borat	bazy@danych.pwr	Bazdy	Danych
8	2222-11-11	13:12:00.000000	2	0	Polski	01:30:00.000000	Władca pierścieni	czwartek@grudzien.edu	Donald	Duck
12	2025-08-12	16:30:00.000000	1	0	Polski	01:30:00.000000	Borat	bazy@danych.pwr	Bazdy	Danych
13	2029-08-28	19:00:00.000000	1	1	Polski	02:32:00.000000	Borat	bazy@danych.pwr	Bazdy	Danych
14	2028-09-12	15:00:00.000000	1	0	Angielski	03:00:00.000000	Borat	bazy@danych.pwr	Bazdy	Danych

Rysunek 33: Widok seansów dla menadżera z poziomu Bazy danych

Wyniki w obu przypadkach są takie same, aczkolwiek lekko zmieniono ich aranżację, jednak dane się pokrywają co oznacza, że program działa tutaj poprawnie.

5.4 Omówienie wybranych rozwiązań programistycznych

5.4.1 Implementacja interfejsu dostępu do bazy danych

Obsługa komunikacji z bazą danych opartą na interfejsie JDBC, opiera się przy pomocy klas znajdujących się w pakiecie dao. Nawiązując połączenie, z bazą danych przy pomocy metody `getCon()` klasy `Mydb`, otwieramy dwustronna komunikację, pomiędzy aplikacją webową, a bazą danych, wykorzystując do tego zmienne klasy `Statement` (do tworzenia i wykonywania poleceń SQL-owych) i `ResultSet` (do pobierania danych z BD). Poniżej przedstawiono przykładową klasę `RachunekDAO.java`, która pobiera dane z widoku `rachunek_view`.

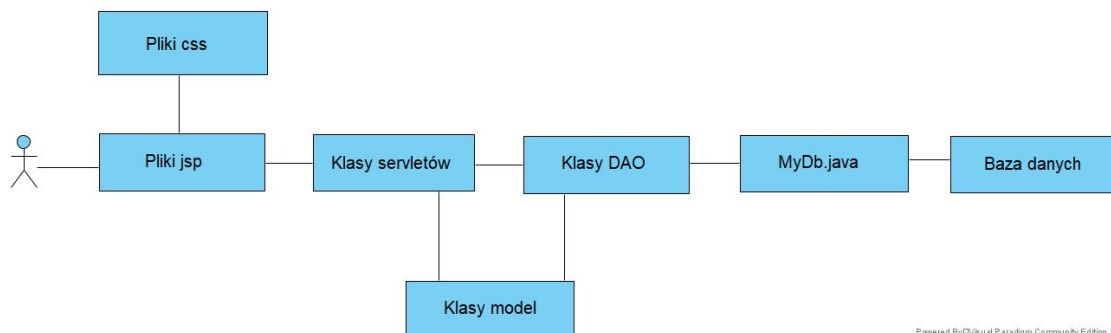
Listing 43: Klasa `RachunekDAO.java`

```
1 public class RachunekDAO {
2
3     Statement stmt ;
4     Statement stmt2 ;
5
6     int id;
7     String data;
8     int kwota;
9     int NIP;
10
11     ResultSet rs;
12     ResultSet rs2;
13
14     public RachunekDAO() {
15
16     }
17     public List<Rachunek> selectAllRachunki(){
18         MyDb db = new MyDb();
19         Connection connection = db.getCon();
20         List<Rachunek> rachunki = new ArrayList<>();
21         try{
22             stmt = connection.createStatement();
23             stmt2 = connection.createStatement();
24
25             rs = stmt.executeQuery("SELECT * FROM bdkino.rachunek_view;");
26             while(rs.next()) {
27                 id = rs.getInt("Id_zakupu");
28                 data = rs.getString("Data");
29                 kwota = rs.getInt("Kwota");
30                 NIP = rs.getInt("NIP");
31                 rachunki.add(new Rachunek(id,data,kwota,NIP));
32             }
33
34         }catch (SQLException e) {
35             // TODO Auto-generated catch block
36             e.printStackTrace();
37         }
38         return rachunki;
39     }
40 }
```

5.4.2 Implementacja wybranych funkcjonalności systemu

Nasza aplikacja webowa składa się z 4 podsekcji plików:

- **Klasy model** przechowujące dane encji w aplikacji.
- **Klasy dao** odpowiedzialne za komunikację pomiędzy interfejsem javy, a bazą danych.
- **Klasy web (Servlety)** odpowiedzialne za komunikację pomiędzy interfejsem javy, a interfejsem webowym, przy pomocy servletów.
- **Pliki jsp oraz css** będące odpowiedzialne za interfejs webowy aplikacji



Rysunek 34: Struktura aplikacji dostępowej.

Klasy web, wywoływane w interfejsie webowym, wywoływanych w plikach .jsp podczas przemieszczania się pomiędzy stronami poleceń ja href=example» lub jaction = example». Za pomocą polecenia @WebServlet("/example"), wywoływany jest odpowiedni klasa web obsługująca odpowiednie zdarzenie i wywołująca odpowiednią metodę klasy dao.

Menadżer, chcący zedytować dane pracownika Mickey Mouse, zamieniając jego imię i nazwisko na Sknerus McKwacz, z panelu menadżera, klika „Zarządzaj pracownikiem”, a następnie „Modyfikuj Dane”. Plik `pracownicyMenagment.jsp`, po kliknięciu hiperłącza uruchomi akcję `PracownikMenagment2`.

Listing 44: Budowa panelu na przykładzie `pracownicyMenagment.jsp`

```
1 <body>
2   <header>
3     <h1 id="mainHeader">Kino Horyzont w</h1>
4   </header>
5
6   <div class="panelMode">
7     <h4 id="panelName">Panel menadżera</h4>
8   </div>
9   <div class="menu">
10    <ul id="menuManager">
11      <li><a href="seansManagement">Zarządzanie seansami</a></li>
12      <li><a href="PracownicyManagement">Zarządzanie pracownikami</a>
13        <</li>
14      <li><a href="FilmManagement">Zarządzanie filmami</a></li>
15    </ul>
16
17    <ul id="menuManager2">
18      <li><a href="RachunekView">Przejdź do rachunku</a></li>
19    </ul>
20
21    <ul id="menuManager3">
22      <li><a href="pracownicyManagementAdd.jsp">Dodaj
23        pracownika</a></li>
24      <li><a href="PracownicyManagement2">Modyfikuj dane</a>
25        <</li>
26    </ul>
27  </div>
```

Akcja zostanie wyłapana, przez Servlet `PracownikServlet2.java`, przy pomocy `@WebServlet("/Manager/PracownicyManagement2")`. Przy pomocy metody `doGet` oraz `listPracownik`, program wywołuje w klasie delegowanej `PracownikDAO` metodę `selectAllPracownik()` zwracającą pełną Listę encji `Pracowników`, w widoku `pracownik_view`, który przesyła do skryptu `pracownicyManagement2.jsp`.

Listing 45: Klasa `PracownikServlet2.java`

```
1 @WebServlet("/Manager/PracownicyManagement2")
2 public class PracownikServlet2 extends HttpServlet{
3     private static final long serialVersionUID = 1L;
4     private PracownikDAO pracownikDAO;
5
6     /**
7      * @see HttpServlet#HttpServlet()
8      */
9     public PracownikServlet2() {
10         this.pracownikDAO=new PracownikDAO();
11     }
12     /**
13      * @see HttpServlet#doPost(HttpServletRequest request,
14      *                          HttpServletResponse response)
15      */
16     protected void doPost(HttpServletRequest request,
17                           HttpServletResponse response) throws ServletException,
18                           IOException {
19         this.doGet(request, response);
20     }
21     /**
22      * @see HttpServlet#doGet(HttpServletRequest request,
23      *                          HttpServletResponse response)
24      */
25     protected void doGet(HttpServletRequest request, HttpServletResponse
26                           response) throws ServletException, IOException {
27         String action = request.getServletPath();
28         try
29         {
30             listPracownik(request, response);
31         } catch (SQLException ex) {
32             throw new ServletException(ex);
33         }
34     }
35
36     private void listPracownik(HttpServletRequest request,
37                               HttpServletResponse response)
38         throws SQLException, IOException, ServletException{
39         List<Pracownik> pracownicy=pracownikDAO.selectAllPracownik();
40         request.setAttribute("listPracownik", pracownicy);
41         RequestDispatcher dispatcher = request.getRequestDispatcher("
42             pracownicyManagement2.jsp");
43         dispatcher.forward(request, response);
44     }
45 }
```

Listing 46: Metoda klasy pracownikDAO selectAllPracownik

```
1  public List<Pracownik> selectAllPracownik(){
2
3      List <Pracownik> pracownicy = new ArrayList<>();
4
5      MyDb db = new MyDb();
6      Connection connection = db.getCon();
7      try{
8          stmt = connection.createStatement();
9          stmt2 = connection.createStatement();
10
11         rs = stmt.executeQuery("select * from pracownik_view");
12         while(rs.next()) {
13             id = rs.getInt("Id_pracownika");
14             imie = rs.getString("Imie");
15             nazwisko = rs.getString("Nazwisko");
16             email = rs.getString("Mail");
17             stanowisko = rs.getString("Stanowisko");
18
19             pracownicy.add(new Pracownik(id,imie,nazwisko,stanowisko,email)
20                 );
21         }
22
23     }catch (SQLException e) {
24         // TODO Auto-generated catch block
25         e.printStackTrace();
26     }
27     return pracownicy;
28 }
```

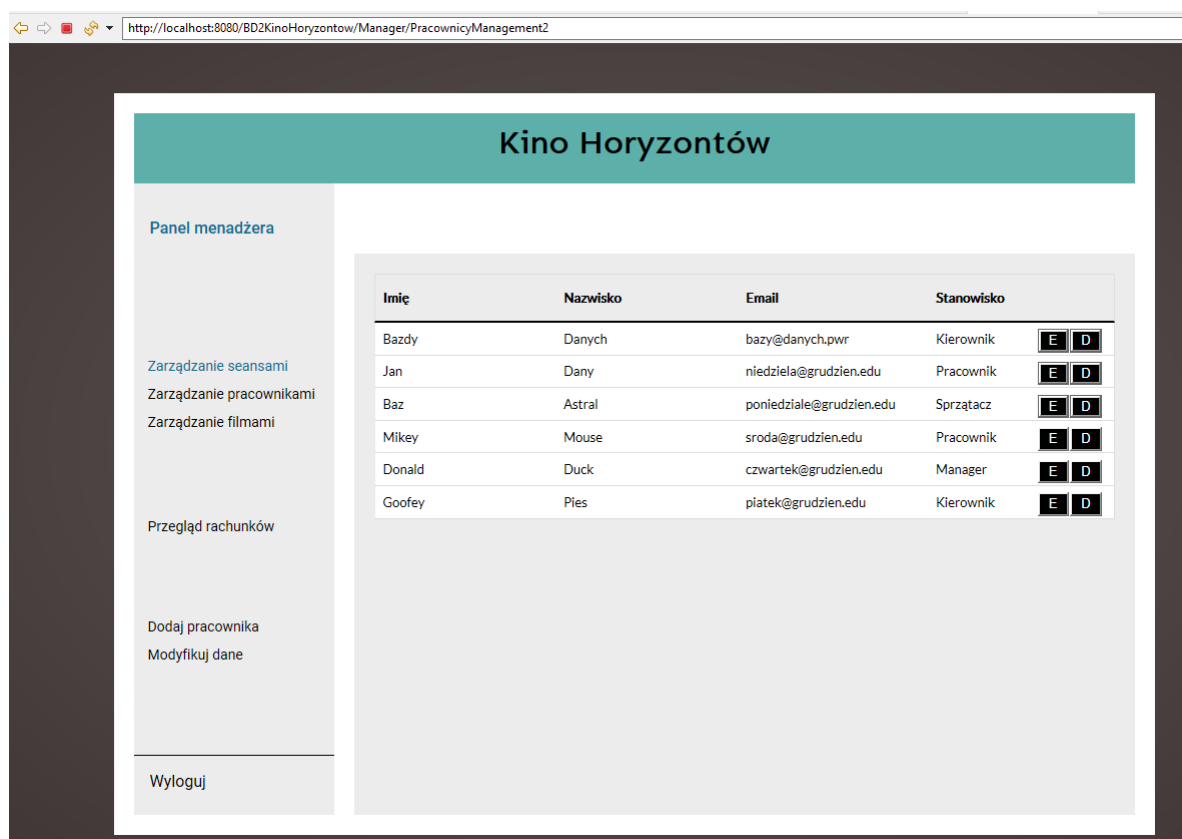
Wykorzystując przesłaną listę, skrypt tworzy tablicę, zawierającą przesłane dane, dodając do niej przyciski D i E, odpowiedzialne za usuwanie i edycje encji. Kliknięcie, przycisku E przy pracowniku Mikey Mouse, spowoduje wywołanie akcji PracownikEdit i przesłaniu do niej wartości id pracowniki Mikey Mouse:

Listing 47: Tabela danych pracownika wywoływana przez skrypt pracownikMenagment2.jsp

```

1 <div class ="tabela">
2     <p>Pracownicy</p>
3
4     <table class="pracownicyTable">
5         <thead>
6             <tr>
7                 <th>Imi </th>
8                 <th>Nazwisko</th>
9                 <th>Email</th>
10                <th id="smallerStanowisko">Stanowisko</th>
11                <th id="btnsED"></th>
12            </tr>
13        </thead>
14        <tbody>
15            <c:forEach var="p" items="${listPracownik}">
16                <tr>
17                    <td><c:out value="${p.getImie()}" /></td>
18                    <td><c:out value="${p.getNazwisko()}" /></td>
19                    <td><c:out value="${p.getEmail()}" /></td>
20                    <td id="smallerStanowisko"><c:out value="${p.
                        getStanowisko()}" /></td>
21
22                    <td id="btnsED">
23
24                        <form id="editBtnlist" action="PracownikEdit" method="post">
25                            <input type="hidden" name="Id_pracownika" value = '${p.getId()
                                }' />
26                            <input type="submit" value="E" /> </form>
27
28                            <form id="deleteBtnList" action="PracownikDelete" method="post"
29                                >
30                                <input type="hidden" name="Id_pracownika" value = '${p.getId()
31                                    }' />
32                                <input type="submit" value="D" /> </form>
33
34                        </td>
35                    </tr>
36                </c:forEach>
37            </tbody>
38        </table>

```



Rysunek 35: Widok z poziomu aplikacji skryptu pracownikMenagment2.jsp

	Id_pracownika	Imie	Nazwisko	Mail	Stanowisko
▶	1	Bazdy	Danych	bazy@danych.pwr	Kierownik
	2	Jan	Dany	niedziela@grudzien.edu	Pracownik
	3	Baz	Astral	poniedziale@grudzien.edu	Sprzątac
	4	Mikey	Mouse	sroda@grudzien.edu	Pracownik
	5	Donald	Duck	czwartek@grudzien.edu	Manager
	6	Goofey	Pies	piatek@grudzien.edu	Kierownik

Rysunek 36: Widok pracownik.view w programie MySql.

Akcja, powoduje uruchomienia servleta PracownikEdit.java, który pobierze ze skryptu id pracownika, pobierze jego pełne dane z BD i wyśle je do skryptu pracownicyManagement-Modify.jsp.

Listing 48: Metoda doGet servleta PracownikEdit.jsp

```

1 protected void doGet(HttpServletRequest request, HttpServletResponse
  response) throws ServletException, IOException {
2     String Ids = request.getParameter("Id_pracownika");
3     int Id = Integer.parseInt(Ids);
4
5
6     Pracownik pracownik = new Pracownik();
7     pracownik = pracownikDAO.getPracownik(Id);
8
9
10    request.setAttribute("pracownik", pracownik);
11    RequestDispatcher dispatcher = request.getRequestDispatcher("
      pracownicyManagementModify.jsp");
12    dispatcher.forward(request, response);
13 }

```

Przy pomocy przesłanych danych, skrypt tworzy okienko edycji danych pracownika, wypełniając je uzyskanymi wcześniej danymi:

Listing 49: Tworzenie formularza do edycji danych w skrypcie pracownicyManagementModify.jsp

```

1 <div class ="pracownikForm">
2     <p>Pracownicy</p>
3     <form id="insertPracownikForm" action="PracownikModify" method=
      "post">
4         <label for="firstName">Imi </label>
5         <input type="text" name="firstName" value="<c:out value='${
      pracownik.getImie()}' />" required/><br/>
6         <label for="lastName">Nazwisko</label>
7         <input type="text" name="lastName" value="<c:out value='${
      pracownik.getNazwisko()}' />" required/><br/>
8         <label for="phoneNum">Stanowisko</label>
9         <input type="text" name="phoneNum" value="<c:out value='${
      pracownik.getStanowisko()}' />" required/><br/>
10        <label for="password">Hasło</label>
11        <input type="password" name="password" value="<c:out value
      ='${pracownik.getHaslo()}' />" required/><br/>
12
13        <input id="addBtn" type="submit" value="Modyfikuj"/>
14
15        <input type="hidden" name="Id_pracownika" value
      ="<c:out value='${pracownik.getId()}' />" /><br/>
16        <!-- <label for="email">Email</label>-->
17        <input type="hidden" name="email" value="<c:out value='${
      pracownik.getEmail()}' />" required/> <br/>
18    </form>
19
20    <form method="get" action="PracownicyManagement">
21        <button id="cancelBtn" type="submit">Anuluj</button>
22    </form>
23
24 </div>

```

The screenshot shows a web browser window with the URL `http://localhost:8080/BD2KinoHoryzontow/Manager/PracownikEdit`. The application has a dark blue header with the title "Kino Horyzontów". On the left, there is a sidebar with the title "Panel menadżera" and several menu items: "Zarządzanie seansami", "Zarządzanie pracownikami", "Zarządzanie filmami", "Przegląd rachunków", and "Wyloguj". The main content area is titled "Pracownicy" and contains a form with the following fields: "Imię" (Name) with the value "Mikey", "Nazwisko" (Surname) with the value "Mouse", "Stanowisko" (Position) with the value "Pracownik", and "Hasło" (Password) which is masked with dots. At the bottom right of the form are two buttons: "Modyfikuj" (Modify) and "Anuluj" (Cancel).

Rysunek 37: Widok z poziomu aplikacji skryptu `pracownicyManagementModify.jsp`

Po wypełnieniu danych i naciśnięciu przycisku „Modyfikuj” zostanie uruchomiony servlet `PracownikModify.java`

The screenshot shows a web browser window with the URL `http://localhost:8080/BD2KinoHoryzontow/Manager/PracownikEdit`. The application is titled "Kino Horyzontów". On the left is a sidebar with the "Panel menadżera" (Manager Panel) containing links for "Zarządzanie seansami" (Manage screenings), "Zarządzanie pracownikami" (Manage employees), "Zarządzanie filmami" (Manage movies), "Przegląd rachunków" (View accounts), and a "Wyloguj" (Logout) button at the bottom. The main content area is titled "Pracownicy" (Employees) and contains a form with the following fields: "Imię" (Name) with the value "Sknerus", "Nazwisko" (Surname) with the value "McKwacz" and a clear button (x), "Stanowisko" (Position) with the value "Pracownik", and "Hasło" (Password) which is masked with dots. To the right of the form are two buttons: "Modyfikuj" (Modify) and "Anuluj" (Cancel).

Rysunek 38: Formularz danych, przed kliknięciem przycisku Modyfikuj

Servlet pobierze wszystkie dane z formularza i przekaże je klasie `PracownikDAO`, która wywoła polecenie SQL-owe updatu pracownika oraz loginu (w przypadku zmiany hasła). Ze względu na przyjęte założenie projektowe, możliwa jest edycja wszystkich danych pracownika, po za email'em jego konta i jego numerem id. Po edycji danych, servlet przeniesie nas na główną stronę zarządzania Pracownikami, gdzie można zobaczyć rezultaty edycji danych

Listing 50: Metoda doGet serwleta PracownikModify.java

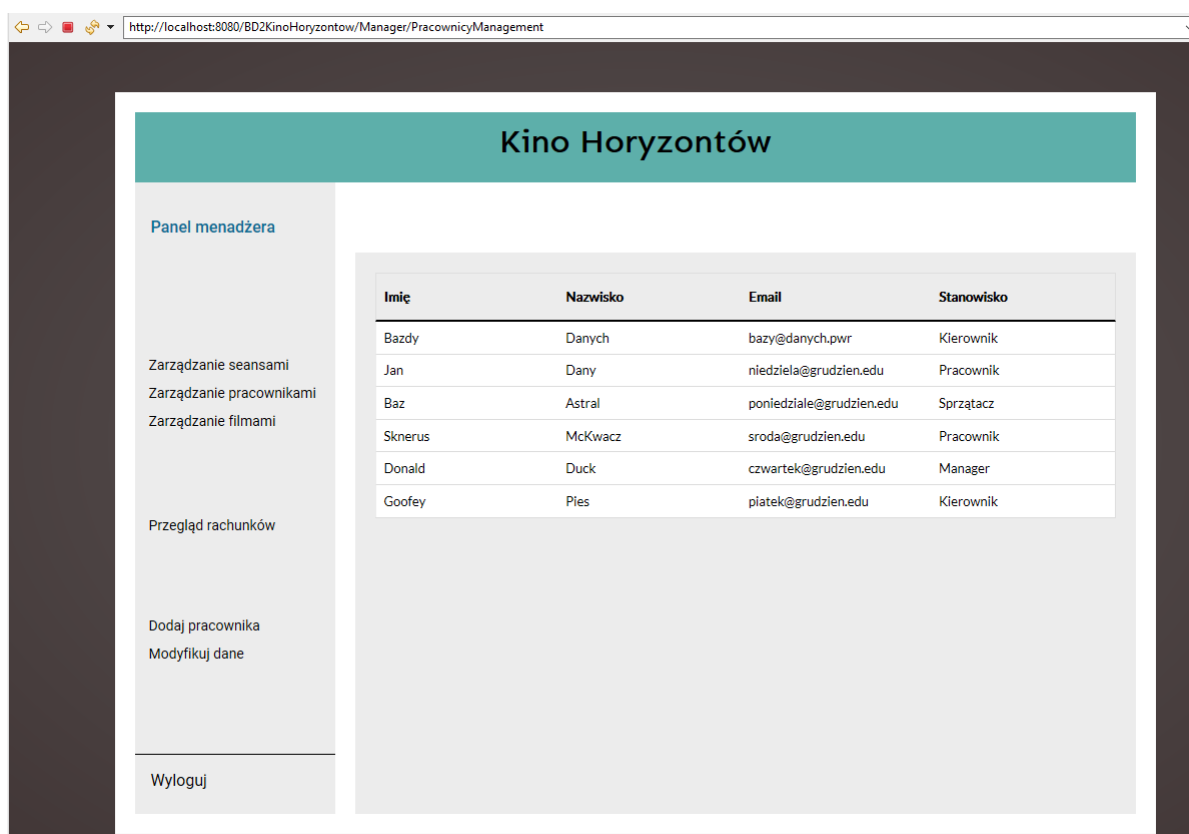
```
1  protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
2      // TODO Auto-generated method stub
3
4      String imie = request.getParameter("firstName");
5      String nazwisko = request.getParameter("lastName");
6      String stanowisko = request.getParameter("phoneNum");
7      String email = request.getParameter("email");
8      String haslo = request.getParameter("password");
9      String Ids = request.getParameter("Id_pracownika");
10     System.out.println(Ids);
11     int id = Integer.parseInt(Ids);
12
13     Pracownik dane = new Pracownik(id, imie, nazwisko, stanowisko, email,
        haslo);
14     PracownikDAO pracownik = new PracownikDAO();
15     try {
16         pracownik.updatePracownik(dane);
17     } catch (SQLException e) {
18         // TODO Auto-generated catch block
19         e.printStackTrace();
20     }
21     response.sendRedirect("PracownicyManagement");
22 }
```

Listing 51: Metoda doGet serwleta PracownikModify.java

```
1  protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
2      // TODO Auto-generated method stub
3
4      String imie = request.getParameter("firstName");
5      String nazwisko = request.getParameter("lastName");
6      String stanowisko = request.getParameter("phoneNum");
7      String email = request.getParameter("email");
8      String haslo = request.getParameter("password");
9      String Ids = request.getParameter("Id_pracownika");
10     System.out.println(Ids);
11     int id = Integer.parseInt(Ids);
12
13     Pracownik dane = new Pracownik(id, imie, nazwisko, stanowisko, email,
        haslo);
14     PracownikDAO pracownik = new PracownikDAO();
15     try {
16         pracownik.updatePracownik(dane);
17     } catch (SQLException e) {
18         // TODO Auto-generated catch block
19         e.printStackTrace();
20     }
21     response.sendRedirect("PracownicyManagement");
22 }
```

Listing 52: Metoda updatePracownik klasy PracownikDAO.java

```
1 public void updatePracownik(Pracownik pracownik) throws SQLException {
2
3     MyDb db = new MyDb();
4     Connection connection = db.getCon();
5     try{
6         stmt = connection.createStatement();
7         id = pracownik.getId();
8         imie = pracownik.getImie();
9         nazwisko = pracownik.getNazwisko();
10        stanowisko = pracownik.getStanowisko();
11        email = pracownik.getEmail();
12        haslo = pracownik.getHaslo();
13
14
15
16        stmt.executeUpdate("update login set Haslo ='"+haslo+"' where
17                               Mail = '"+email+"' ;");
18
19        stmt.executeUpdate("update pracownik set Imie ='"+imie+"',
20                               Nazwisko='"+nazwisko+"', Stanowisko = '"+stanowisko+"',
21                               where Id_pracownika = '"+id+"' ;");
22    }catch (SQLException e) {
23        // TODO Auto-generated catch block
24        e.printStackTrace();
25    }
```



Rysunek 39: Efekt operacji edycji na wywołanym skrypcie `pracownicyManagement.jsp`

	Id_pracownika	Imie	Nazwisko	Mail	Stanowisko
▶	1	Bazdy	Danych	bazy@danych.pwr	Kierownik
	2	Jan	Dany	niedziela@grudzien.edu	Pracownik
	3	Baz	Astral	poniedziale@grudzien.edu	Sprzątac
	4	Sknerus	McKwacz	sroda@grudzien.edu	Pracownik
	5	Donald	Duck	czwartek@grudzien.edu	Manager
	6	Goofey	Pies	piatek@grudzien.edu	Kierownik
*	NULL	NULL	NULL	NULL	NULL

Rysunek 40: Tablica `Pracownik` w MySQL, po edycji danych z poziomu aplikacji.

5.4.3 Implementacja mechanizmów bezpieczeństwa

W ramach aplikacji zaimplementowano mechanizm logowania. Do zalogowania wymagane jest podanie emailu i hasła. Dane te następnie porównywane są z istniejącymi w bazie danych i zwracają poziom uprawnień danego użytkownika (jeżeli dane w ogóle istnieją, jeżeli nie istnieją osoba logująca się zwracana jest do ekranu logowania). Na podstawie uzyskanego poziomu uprawnień (w bazie nazwane jest to `Id` typu) użytkownik aplikacji przenoszony jest na odpowiednią stronę adekwatną do swoich uprawnień. Najprawdopodobniej nie wszystkie mechanizmy bezpieczeństwa zostały zastosowane poprawnie, ze względu na co najwyżej przeciętną wiedzę, na temat tworzenia aplikacji webowych w Javie, posiadaną przez jej twórców.

6. Podsumowanie i wnioski

Podczas zadania projektowego, utwierdzono i pogłębiono wiedzę z zakresu projektowania systemów baz danych. Zaprojektowano ją od zera, na podstawie opisu słownego, projektując wstępne encje i ustalając jej wstępny rozmiar oraz planowane technologie. Kończącym etapem projektowania bazy, było utworzenie modelu fizycznego bazy danych, wraz z mechanizmami przetwarzania danych oraz projekt aplikacji, z wykorzystaniem makiety interfejsu graficznego oraz diagramu przypadku użycia. Utworzono bazę danych z wbudowanymi mechanizmami triggerów, widoków, indeksów, sekwencji oraz procedur składowych, znacznie usprawniającą jej działanie na etapie tworzenia aplikacji. Najwięcej problemów napotkano, na etapie implementacji aplikacji webowej, poznając działanie Servletów, mechanizmów komunikacji pomiędzy interfejsami webowymi, javy, BD oraz podstaw CSS-a. Finalny efekt, może zawierać wiele błędów, a sama wielkość projektu okazała się w niewielkim stopniu przytłaczająca dla projektantów, jednak udało się zrealizować praktycznie wszystkie funkcjonalności, przyjęte w fazie projektowania aplikacji.

7. Kod źródłowy

Kod źródłowy aplikacji oraz bazy danych umieszczony został na platformie *GitHub*.

Aplikacja i skrypt SQL:

<https://github.com/plebian1/Bazy-Danych-2-Kino-Final>

Literatura

- [1] Visual Paradigm: *How to Model Relational Database Design with ERD* - <https://www.visual-paradigm.com/tutorials/how-to-model-relational-database-with-erd.jsp>
- [2] Oracle : *MySQL Documentation* <https://dev.mysql.com/doc/>
- [3] Java Guides: *Tworzenie aplikacji webowych w języku Java* <https://www.youtube.com/watch?v=9ToWzrRihv4>
- [4] CodeJava.net *Mechanizm logowania do aplikacji* <https://www.codejava.net/coding/how-to-code-login-and-logout-with-java-servlet-jsp-and-mysql>
- [5] jinu jawad m: *Export mysql database as .sql* <https://www.youtube.com/watch?v=Fls120oEv-k&t=127s>