Bazy danych

Projekt zaliczeniowy

Baza danych multikina

Natalia Kramarz

1 Podstawowe założenia projektu

Projekt umożliwia podstawową obsługę multikina:

- **obsługę sprzedaży biletów**, która polega na utworzeniu rezerwacji przez klienta lub pracownika, dodaniu odpowiedniej ilości biletów przypisanej do danej rezerwacji oraz zarezerwowaniu poszczególnych miejsc w sali kinowej na dany seans.
- **generowanie raportów dotyczących sprzedaży biletów oraz przychodów generowanych przez każdy film**
- **dodawanie tytułów filmów do repertuaru**
- **tworzenie seansów**

Przechowuje on dane:

- **Klientów, pracowników** - imię, nazwisko, w przypadku klientów kod pocztowy, potrzebny do statystyk
- **Filmów** - tytuł, rok produkcji, czas trwania oraz gatunek
- **Seansów** - data seansu, identyfikator filmu, który będzie wyświetlany oraz numer sali kinowej, w której seans się odbędzie
- **Sal kinowych** - ilość miejsc w poszczególnej sali kinowej
- **Rezerwacji** - ile biletów zostało sprzedanych, czy zakupu dokonano online czy też na miejscu, dla każdego biletu wchodzącego w skład rezerwacji jest przypisywane wybrane miejsce w sali kinowej
- **Cen biletów** - baza umożliwia zmianę cen biletów w czasie oraz przechowuje starsze ceny biletów

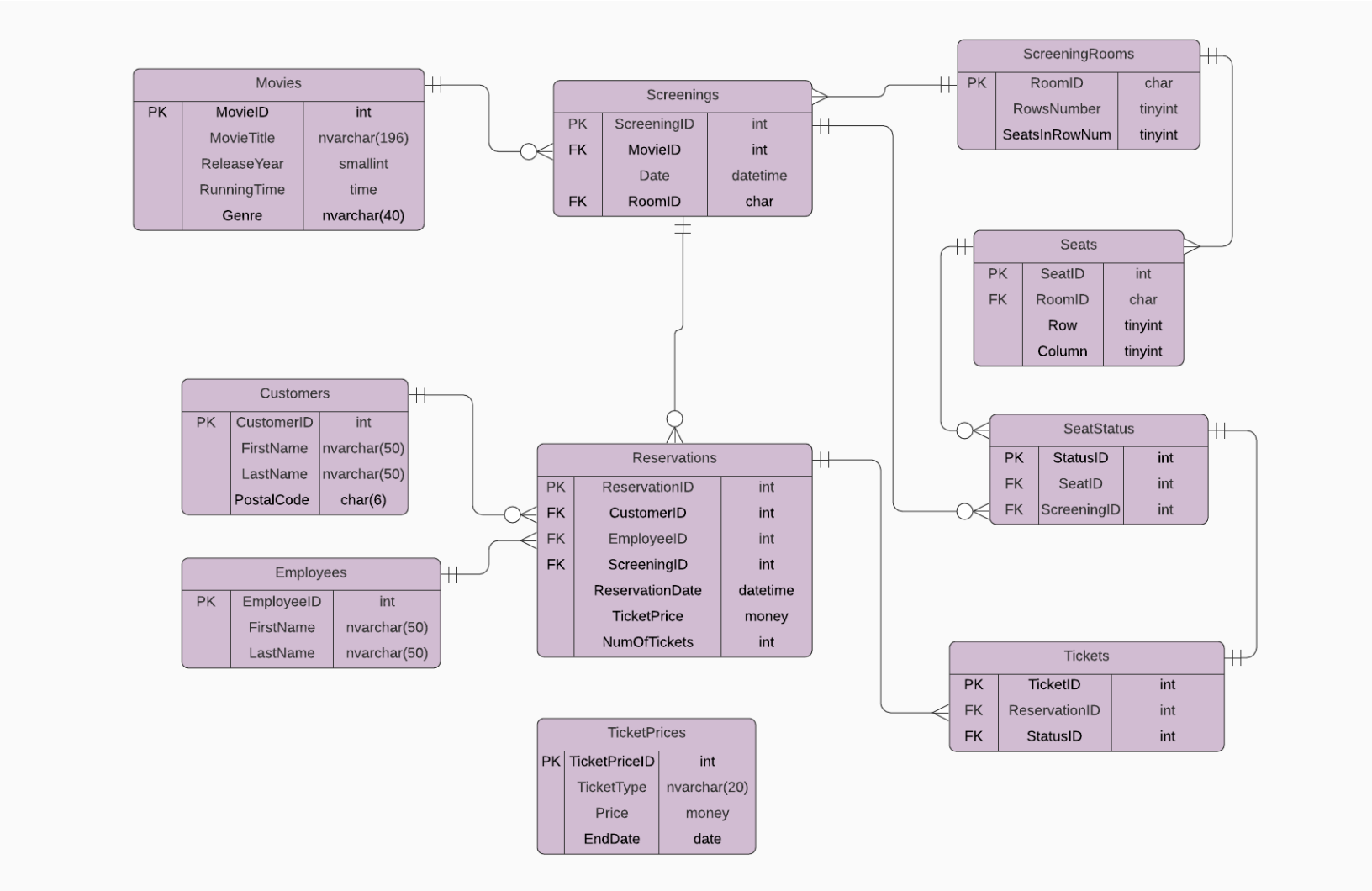
2 Ograniczenia przyjęte przy projektowaniu

- Nie można usunąć ani dodać żadnej z sal kiniowych, ani żadnego fotela w poszczególnych salach kinowych
- Nie ma dwóch filmów o tych samych tytułach
- Nie można zarezerwować więcej niż raz danego miejsca dla tego samego seansu

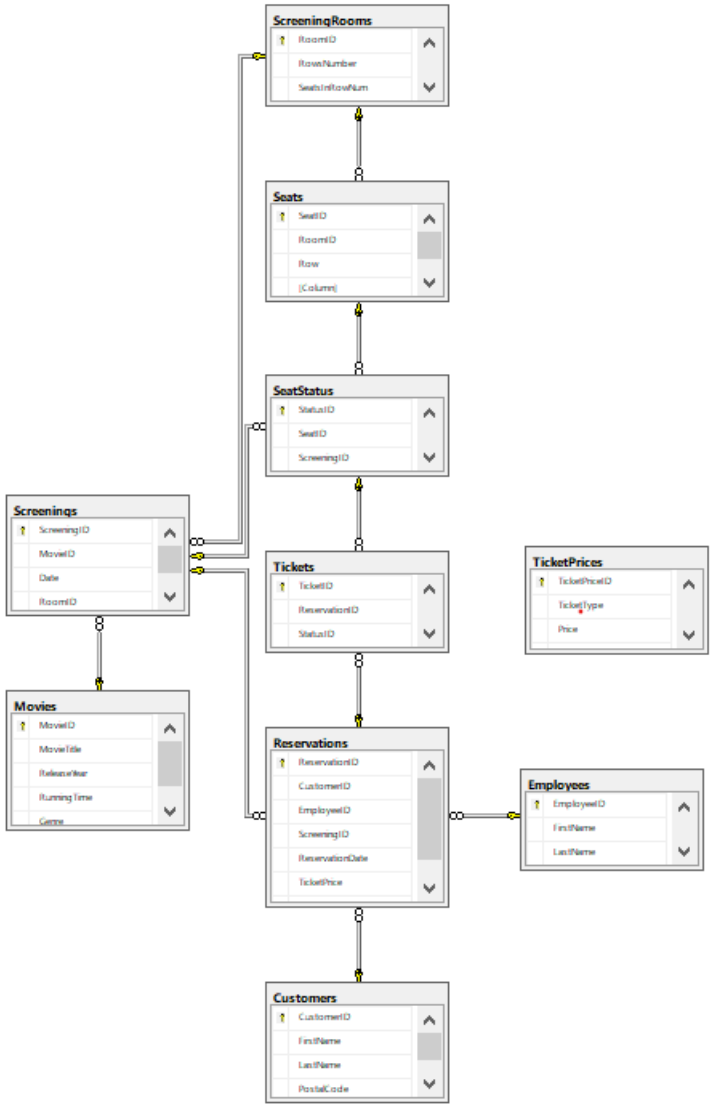
3 Dodatkowe więzy integralności

- **MovieNameLength** - sprawdza czy tytuł filmu ma długość od 2 do 196 znaków
- **CHK_Year** - sprawdza czy rok produkcji filmu jest pomiędzy rokiem 1900, a obecną datą + 5 lat
- **RowsNumber** - sprawdza czy w rzędzie jest od 0 do 30 krzeseł
- **CHKNamesCustomers, CHKNamesEmployees** - sprawdzają poprawność pisowni imienia i nazwiska
- **TicketTypeLength** - sprawdza czy nazwa biletu jest dłuższa niż jeden znak
- **CHCK_Name** - sprawdza czy tytuł filmu różni się od innych tytułów
- **CHCK_SeatScreening** - sprawdza czy istnieje już rezerwacja miejsca na dany seans

4 Diagram ER


Movies
PK MovieID int
MovieTitle nvarchar(196)
ReleaseYear smallint
RunningTime time
Genre nvarchar(40)
Screenings
PK ScreeningID int
FK MovieID int
Date datetime
FK RoomID char
ScreeningRooms
PK RoomID char
RowsNumber tinyint
SeatsInRowNum tinyint
Seats
PK SeatID int
FK RoomID char
Row tinyint
Column tinyint
Customers
PK CustomerID int
FirstName nvarchar(50)
LastName nvarchar(50)
PostalCode char(6)
Reservations
PK ReservationID int
FK CustomerID int
FK EmployeeID int
FK ScreeningID int
ReservationDate datetime
TicketPrice money
NumOfTickets int
SeatStatus
PK StatusID int
FK SeatID int
FK ScreeningID int
Employees
PK EmployeeID int
FirstName nvarchar(50)
LastName nvarchar(50)
Tickets
PK TicketID int
FK ReservationID int
FK StatusID int
TicketPrices
PK TicketPriceID int
TicketType nvarchar(20)
Price money
EndDate date

5 Schemat bazy danych



6 Funkcje

6.1 Funkcja find_screenings

Funkcja przyjmuje tytuł filmu oraz datę, od której ma znaleźć seanse danego filmu i następnie zwraca tabelę seansów spełniających kryteria

```
GO
CREATE FUNCTION [dbo].find_screenings(@movieTitle nvarchar(256), @date datetime = NULL)
RETURNS @screenings_tab TABLE (ScreeningID int, MovieID int, [Date] datetime, RoomID char)
BEGIN
    if (@date IS NULL)
      SET @date = GETDATE()
    declare @movie_id int = (SELECT [dbo].find_movie(@movieTitle))
    INSERT INTO @screenings_tab(ScreeningID, MovieID, [Date], RoomID)
     SELECT * FROM Screenings WHERE  MovieID = @movie_id AND DATEDIFF(day, @date, [Date]) >= 0
    return
END
GO
```

6.2 Funkcja find_free_seats

Funkcja przyjmuje id seansu i zwraca listę wolnych miejsc dla danego seansu

```
GO
CREATE FUNCTION [dbo].find_free_seats(@screeningID int)
RETURNS @seatsTab TABLE (SeatID int)
BEGIN
  declare @roomID char
  SELECT @roomID = RoomID FROM SCREENINGS WHERE ScreeningID = @screeningID
  INSERT INTO @seatsTab
    SELECT SeatID FROM Seats WHERE @roomID = RoomID and NOT EXISTS (SELECT SeatID FROM SeatStatus WHERE ScreeningID =
   @screeningID)
  return
END
GO
```

6.3 Funkcja find_movie

Funkcja przyjmuje tytuł filmu i zwraca jego id

```
GO
CREATE FUNCTION [dbo].find_movie(@movieTitle nvarchar(256)) RETURNS int
BEGIN
  return (SELECT MovieID FROM Movies WHERE MovieTitle = @movieTitle)
END
GO
```

6.4 Funkcja find_ticket_price

Funkcja przyjmuje datę (jeżeli data jest pusta, to ustawia ją na aktualną datę) i na jej podstawie sprawdza jaka będzie cena biletu ze względu na dzień tygodnia

```
GO
CREATE FUNCTION [dbo].find_ticket_price(@date datetime = NULL)
RETURNS money
BEGIN
  declare @weekday int
  if (@date IS NULL)
      SET @date = GETDATE()
  SELECT  @weekday = DATEPART(WEEKDAY, @date)
  if @weekday = 0
    return (SELECT PRICE FROM TicketPrices WHERE TicketType = 'monday' AND EndDate IS NULL)
  if @weekday = 2
    return (SELECT PRICE FROM TicketPrices WHERE TicketType = 'wednesday' AND EndDate IS NULL)

  return (SELECT PRICE FROM TicketPrices WHERE TicketType = 'regular' AND EndDate IS NULL)
END
GO
```

6.5 Funkcja employee_who_sold_most_tickets_week

Funkcja zwraca id pracownika, który sprzedał najwięcej biletów w ciągu 7 dni od podanej daty

```
GO
```

```
CREATE FUNCTION [dbo].employee_who_sold_most_tickets_week(@date date = NULL) RETURNS INT
BEGIN
  IF (@date IS NULL)
    SET @date = GETDATE()
  declare @empId int
  SELECT TOP 1 @empID = Employees.EmployeeID FROM Employees
  JOIN Reservations
  ON Reservations.EmployeeID IS NOT NULL AND Reservations.EmployeeID = Employees.EmployeeID
  AND ReservationDate >= DATEADD(day,-7, @date)
  GROUP BY Employees.EmployeeID
  ORDER BY(SUM(TicketPrice * NumOfTickets))

  return @empID
END
GO
```

7 Procedury składowane

7.1 Procedura make_reservation

Procedura umożliwia zrobienie rezerwacji - jeżeli jest dokonywana online to przypisywany jest jej numer klienta, a jeżeli na miejscu w kinie to przypisywany jest jej numer pracownika, który dokonał rezerwacji. Aby można było przypisać większą ilość biletów do rezerwacji jest zdefiniowany dodatkowy typ danych:

```
CREATE TYPE SeatsTableType
AS TABLE (SeatsID int)
```

Procedura zlicza liczbę miejsc do zarezerowania, znajduje cenę biletu, która zależy od dnia tygodnia, służy do tego funkcja find_ticket_price, gdy zostanie utworzona rezerwacja, do tabelki SeatStatus dodawane są rekordy z informacją, że dla tego seansu zajęte są miejsca z rezerwacji. Na podstawie identyfikatorów statusów foteli są następnie tworzone bilety.

Jako, że wiele osób może próbować rezerwować te same miejsca w tym samym momencie, to jest ustawiony XACT_ABORT ON, więc jeżeli któraś z operacji w transakcji nie powiodzie się, to zostaną wycofane wszystkie operacje.

```
GO
CREATE PROC make_reservation(@SeatsTab SeatsTableType READONLY, @screeningID int, @EmployeeID int = NULL, @CustomerID
    int = NULL)
AS
    BEGIN TRAN
```

```
    SET XACT_ABORT ON
    declare @ticketPrice money
    declare @numOfTickets int
    SELECT @numOfTickets = COUNT(DISTINCT SeatsID) FROM @SeatsTab
    SELECT @ticketPrice = [dbo].find_ticket_price(default)
    INSERT INTO Reservations (CustomerID, EmployeeID, ScreeningID, ReservationDate, TicketPrice, NumOfTickets) VALUES
    (@CustomerID, @EmployeeID, @screeningID, GETDATE(), @ticketPrice, @numOfTickets)
    declare @reservationID int
    SELECT @reservationID = SCOPE_IDENTITY()
    DECLARE @StatusIds TABLE (id int)
    INSERT INTO SeatStatus
      OUTPUT INSERTED.StatusID INTO @StatusIds
      SELECT *, @screeningID FROM @SeatsTab
    INSERT INTO Tickets
      SELECT @reservationID, id FROM @StatusIds
    COMMIT
GO
```

7.2 Procedura seats_insertion

Procedura wstawia do tabeli Seats ilość foteli zadeklarowaną dla danej sali kinowej oraz przypisuje każdemu z foteli numer odpowiedniej sali kinowej, numer rzędu oraz kolumny, w której dany fotel się znajduje

```
GO
CREATE PROCEDURE seats_insertion(@RoomID char, @RowsNumber tinyint, @SeatsInRow tinyint) AS BEGIN
  DECLARE @i tinyint = 1
  DECLARE @j tinyint = 1
  WHILE @i <= @RowsNumber
  BEGIN
    SET @j = 1
    WHILE @j <= @SeatsInRow
    BEGIN
      INSERT INTO Seats(RoomID, [Row], [Column])
      VALUES (@RoomID, @i, @j)
      SET @j += 1
    END
    SET @i += 1
  END
```

```
END
GO
```

7.3 Procedura seats_insertion_rooms

Procedura wywołuje dla wszystkich sal kinowych wpisanych do tabeli ScreeningRooms procedurę wstawiania foteli do tabeli Seats

```
GO
CREATE PROCEDURE seats_insertion_rooms AS BEGIN
DECLARE @RoomID char
SELECT TOP 1 @RoomID=RoomID FROM ScreeningRooms ORDER BY RoomID
DECLARE @RowsNumber tinyint
DECLARE @SeatsInRow tinyint
DECLARE @i int
SELECT @i = COUNT(*) FROM ScreeningRooms
WHILE @i > 0
BEGIN
  SELECT @RowsNumber = RowsNumber, @SeatsInRow = SeatsInRowNum FROM ScreeningRooms WHERE @RoomID = RoomID
  EXECUTE seats_insertion @RoomID, @RowsNumber,  @SeatsInRow
  PRINT @RoomID
  SELECT TOP 1 @RoomID=RoomID FROM ScreeningRooms WHERE RoomID > @RoomID ORDER BY RoomID
  SET @i -= 1
END
END
GO
```

7.4 Procedura change_ticket_price

Procedura umożliwia menadżerowi kina zmianę cen biletów.

```
GO
CREATE PROC change_ticket_price(@employeeID int, @ticketType nvarchar(20),  @price money,  @date date = NULL)
AS
BEGIN
  IF(@employeeID IS NOT NULL AND @employeeID = 1)
  BEGIN
    if (@date IS NULL)
      SET @date = GETDATE()
```

```
    UPDATE TicketPrices
    SET EndDate = DATEADD(day, -1, @date)
    WHERE @ticketType = TicketType AND EndDate IS NULL
    INSERT INTO TicketPrices VALUES
      (@ticketType, @price, NULL)
  END
  ELSE BEGIN
    print 'Only cinema manager can change ticket prices!'
  END
END
```

7.5 Procedura add_movie

Procedura pozwala tylko menadżerowi kina na dodanie nowego filmu do repertuaru oraz sprawdza czy gatunek filmu jest wpisany.

```
GO
CREATE PROC add_movie(@employeeID int, @MovieTitle nvarchar(196), @ReleaseYear smallint, @RunningTime time,  @genre
    nvarchar(40))
AS
BEGIN
  IF(@employeeID IS NOT NULL AND @employeeID = 1)
  BEGIN
    IF (@genre IS NOT NULL)
    BEGIN
      INSERT INTO Movies VALUES
        (@MovieTitle, @ReleaseYear, @RunningTime, @genre)
    END
    ELSE BEGIN
      print('Invalid parameters!')
    END
  END
  ELSE BEGIN
    print 'Only cinema manager can add movies to cinema program!'
  END
END
GO
```

8 Widoki

8.1 Widok FilmsEarnings

Widok wyświetla listę nazw filmów wraz z przychodami wygenerowanymi przez każdy z filmów.

```
CREATE View FilmsEarnings AS
SELECT M.MovieTitle, ISNULL(SUM(TicketPrice * NumOfTickets), 0) [Total Earnings] FROM Movies M
LEFT JOIN Screenings S  ON M.MovieID = S.MovieID
LEFT JOIN Reservations R ON S.ScreeningID = R.ScreeningID
GROUP BY M.MovieID, M.MovieTitle
```

8.2 Widok MonthIncome

Widok zwraca listę przychodów wygenerowanych ze sprzedaży biletów w poszczególnych miesiącach.

```
CREATE VIEW MonthIncome AS
SELECT DATEPART(year,ReservationDate) AS [Year],
DateName(month, DateAdd(month, DATEPART(month, ReservationDate), -1 )) AS [Month], SUM(TicketPrice * NumOfTickets)
AS [Total Earnings] FROM Reservations
GROUP BY DATEPART(year, ReservationDate), DATEPART(month, ReservationDate)
```

8.3 Widok EarningsPostalCode

Widok wyświetla informację o sprzedaży biletów z pogrupowaniem ze względu na kody pocztowe.

```
CREATE VIEW EarningsPostalCode AS
SELECT PostalCode, ISNULL(SUM(TicketPrice * NumOfTickets), 0) AS [Total Earnings] FROM Customers C
LEFT JOIN Reservations R
ON R.CustomerID IS NOT NULL AND R.CustomerID = C.CustomerID
GROUP BY PostalCode
```

8.4 Widok StartEndTimeScreening

Widok wyświetla id seansu wraz z jego datą rozpoczęcia i zakończenia oraz numerem sali, w której się odbywa.

```sql
CREATE VIEW StartEndTimeScreening
AS
SELECT ScreeningID, [DATE] AS StartTime, [Date] + CAST(RunningTime AS datetime) AS EndTime, RoomID
FROM Screenings
JOIN Movies
ON Movies.MovieID = Screenings.MovieID
```

8.5 Widok DayIncome

Widok zwraca listę dni oraz przychód ze sprzedaży biletów w poszczególnych dniach.

```sql
CREATE VIEW DayIncome
AS
SELECT CONVERT(date, ReservationDate) AS [Date], SUM(TicketPrice * NumOfTickets) AS [Total Earnings] FROM Reservations
GROUP BY
CONVERT(date, ReservationDate)
```

9 Wyzwalacze

9.1 Wyzwalacz tr_screening_insert

Wyzwalacz sprawdza czy istnieją seanse, których numer sali kinowej, godziny rozpoczęcia i zakończenia nachodzą się, z którymś z dodawanych seansów, jeżeli istnieją, to nowe seanse nie są dodawane

```sql
GO
CREATE TRIGGER tr_screening_insert ON Screenings
INSTEAD OF INSERT
AS
  IF NOT EXISTS (SELECT INSERTED.[DATE] FROM INSERTED JOIN StartEndTimeScreening ON INSERTED.[DATE] >= StartTime AND
   INSERTED.[DATE] <= EndTime AND Inserted.RoomID = StartEndTimeScreening.RoomID)
     AND NOT EXISTS(SELECT INSERTED.[DATE] FROM INSERTED WHERE [DATE] <= GETDATE())
  BEGIN

    INSERT INTO Screenings(MovieID, [Date], RoomID)
      SELECT MovieID, [Date], RoomID FROM Inserted
  END
GO
```

9.2 Wyzwalacz tr_screening_rooms_insert_delete

Wyzwalacz zapobiega usunięciu którejś z sal kinowych

```
GO
CREATE TRIGGER tr_screening_rooms_insert_delete ON ScreeningRooms
INSTEAD OF INSERT, DELETE
AS
  PRINT ('Cannot delete or add a new screening room')
GO
```

9.3 Wyzwalacz tr_seats_insert_delete

Wyzwalacz zapobiega usunięciu foteli z sal kinowych

```
GO
CREATE TRIGGER tr_seats_insert_delete ON Seats
INSTEAD OF INSERT, DELETE
AS
  PRINT ('Cannot delete or add new seats')
GO
```

9.4 Wyzwalacz tr_tickets_delete

Wyzwalacz przy usuwaniu biletu zwalnia również rezerwację miejsca

```
GO
CREATE TRIGGER tr_tickets_delete ON Tickets
INSTEAD OF DELETE
AS
  DELETE FROM Tickets WHERE TicketID IN (SELECT TicketID FROM DELETED)
  DELETE FROM SeatStatus WHERE StatusID IN (SELECT StatusID FROM DELETED)
GO
```

9.5 Wyzwalacz tr_reservations_delete

Wyzwalacz usuwa rezerwację oraz wszystkie przypisane do niej bilety

```
GO
CREATE TRIGGER tr_reservations_delete ON Reservations
INSTEAD OF DELETE
AS
  DELETE FROM Tickets WHERE ReservationID IN (SELECT ReservationID FROM DELETED)
  DELETE FROM Reservations WHERE ReservationID IN (SELECT ReservationID FROM DELETED)
GO
```

10 Skrypt tworzący bazę danych

```
IF OBJECT_ID('Movies', 'U') IS NOT NULL DROP TABLE Movies

CREATE TABLE Movies (
  MovieID int PRIMARY KEY IDENTITY(0,1),
  MovieTitle nvarchar(196) NOT NULL,
  ReleaseYear smallint NOT NULL,
  RunningTime time NOT NULL,
  Genre nvarchar(40),
  CONSTRAINT MovieNameLength CHECK (LEN(MovieTitle) >= 2 AND LEN(MovieTitle) <= 196),
  CONSTRAINT CHK_Year CHECK (ReleaseYear >= 1900 AND ReleaseYear <= (YEAR(getdate()) + 5)),
  CONSTRAINT CHCK_Name UNIQUE(MovieTitle)
);


IF OBJECT_ID('ScreeningRooms', 'U') IS NOT NULL DROP TABLE ScreeningRooms
CREATE TABLE ScreeningRooms (
  RoomID char PRIMARY KEY,
  RowsNumber tinyint CHECK(RowsNumber > 0 and RowsNumber <= 30) NOT NULL,
  SeatsInRowNum tinyint DEFAULT 20,
);

IF OBJECT_ID('Seats', 'U') IS NOT NULL DROP TABLE Seats
CREATE TABLE Seats (
  SeatID int PRIMARY KEY IDENTITY(1,1),
  RoomID char NOT NULL FOREIGN KEY REFERENCES ScreeningRooms(RoomID),
  [Row] tinyint,
  [Column] tinyint,
```

```
);

IF OBJECT_ID('Customers', 'U') IS NOT NULL DROP TABLE Customers
CREATE TABLE Customers (
  CustomerID int PRIMARY KEY IDENTITY(0,1),
  FirstName nvarchar(50) NOT NULL,
  LastName nvarchar(50) NOT NULL,
  PostalCode char(6),
  Constraint CHKNamesCustomers CHECK ((FirstName NOT LIKE '%[-!#%&+,./:;<=>@`{|}~"()*\\\_\^\?\[\]\'']%' {ESCAPE '\'})
   OR (LastName NOT LIKE '%[-!#%&+,./:;<=>@`{|}~"()*\\\_\^\?\[\]\'']%' {ESCAPE '\'}))
);

IF OBJECT_ID('Employees', 'U') IS NOT NULL DROP TABLE Employees
CREATE TABLE Employees (
  EmployeeID int PRIMARY KEY IDENTITY(1,1),
  FirstName nvarchar(50) NOT NULL,
  LastName nvarchar(50) NOT NULL,
  Constraint CHKNamesEmployees CHECK ((FirstName NOT LIKE '%[-!#%&+,./:;<=>@`{|}~"()*\\\_\^\?\[\]\'']%' {ESCAPE '\'})
   OR (LastName NOT LIKE '%[-!#%&+,./:;<=>@`{|}~"()*\\\_\^\?\[\]\'']%' {ESCAPE '\'}))
);

IF OBJECT_ID('TicketPrices', 'U') IS NOT NULL DROP TABLE TicketPrices
CREATE TABLE TicketPrices (
  TicketPriceID int PRIMARY KEY IDENTITY(1,1),
  TicketType nvarchar(20) NOT NULL,
  Price money NOT NULL,
  EndDate date,
  CONSTRAINT TicketTypeLength CHECK (LEN(TicketType) >= 2)
);

IF OBJECT_ID('Screenings', 'U') IS NOT NULL DROP TABLE Screenings
CREATE TABLE Screenings (
  ScreeningID int PRIMARY KEY IDENTITY(1,1),
  MovieID int FOREIGN KEY REFERENCES Movies(MovieID),
  [Date] datetime NOT NULL,
  [RoomID] char NOT NULL FOREIGN KEY REFERENCES ScreeningRooms(RoomID),
);

IF OBJECT_ID('Reservations', 'U') IS NOT NULL DROP TABLE Reservations
```

```
CREATE TABLE Reservations (
  ReservationID int PRIMARY KEY IDENTITY(1,1),
  CustomerID int FOREIGN KEY REFERENCES Customers(CustomerID),
  EmployeeID int FOREIGN KEY REFERENCES Employees(EmployeeID),
  ScreeningID int NOT NULL FOREIGN KEY REFERENCES Screenings(ScreeningID),
  ReservationDate datetime NOT NULL,
  TicketPrice money,
  NumOfTickets int CHECK (NumOfTickets BETWEEN 0 AND 280)
);

IF OBJECT_ID('SeatStatus', 'U') IS NOT NULL DROP TABLE SeatStatus
CREATE TABLE SeatStatus (
  StatusID int PRIMARY KEY IDENTITY(1,1),
  SeatID int NOT NULL FOREIGN KEY REFERENCES Seats(SeatID),
  ScreeningID int NOT NULL FOREIGN KEY REFERENCES Screenings(ScreeningID),
  CONSTRAINT CHCK_SeatScreening UNIQUE(SeatID, ScreeningID)
);

IF OBJECT_ID('Tickets', 'U') IS NOT NULL DROP TABLE Tickets
CREATE TABLE Tickets (
  TicketID int PRIMARY KEY IDENTITY(1,1),
  ReservationID int NOT NULL FOREIGN KEY REFERENCES Reservations(ReservationID),
  StatusID int NOT NULL FOREIGN KEY REFERENCES SeatStatus(StatusID)
);
```

11 Typowe zapytania

```
SELECT FirstName, LastName, PostalCode FROM Customers
SELECT FirstName, LastName FROM Employees
SELECT MovieTitle, ReleaseYear, RunningTime, Genre FROM Movies

SELECT * FROM Screenings WHERE [Date] <= DATEADD(day, 7, GETDATE())

SELECT MovieTitle, [Date] FROM Movies
JOIN Screenings
ON Movies.MovieID = Screenings.MovieID
WHERE [Date] <= DATEADD(day, 7, GETDATE())
```

```
11
12 SELECT MovieTitle, ReleaseYear, RunningTime FROM Movies WHERE GENRE = 'Thriller'
13
14 SELECT * FROM MonthIncome ORDER BY [Total Earnings] DESC
15
16 SELECT Screenings.ScreeningID, ISNULL(SUM(NumOfTickets), 0) [Number of sold tickets] FROM Screenings
17 LEFT JOIN Reservations
18 ON Screenings.ScreeningID = Reservations.ScreeningID
19 GROUP BY Screenings.ScreeningID
20 ORDER BY COUNT(ReservationID) DESC
```

12 Strategia pielęgnacji bazy danych

Procedura tworząca kopię zapasową bazy danych:

```
1 GO
2 CREATE PROCEDURE backup_multikino
3 AS
4 BACKUP DATABASE [Multikino]
5 TO DISK = N'C:\temp\Multikino.bak'
6 WITH NOFORMAT, NOINIT,
7 NAME = N'Multikino Database Backup', SKIP, NOREWIND, NOUNLOAD,  STATS = 10
8 GO
```

Przywracanie bazy danych:

```
1 USE [master]
2 RESTORE DATABASE [Multikino]
3 FROM DISK = N'C:\temp\Multikino.bak' WITH  FILE = 1,  NOUNLOAD,  STATS = 5
4 GO
```

Aby przyśpieszyć wyszukiwanie statusów miejsc w sali kiniowej trzeba regularnie usuwać bilety i statusy miejsc z seansów, które już się odbyły, np. po siedmiu dniach od danego seansu. Można to robić następującą kwerendą:

```
1 DELETE FROM Tickets WHERE TicketID IN
2 (SELECT TicketID FROM Tickets T JOIN Reservations R ON T.ReservationID = R.ReservationID JOIN Screenings S ON S.
     ScreeningID = R.ScreeningID AND DATEDIFF(day, GETDATE(), S.[Date]) > 7)
```

13 Skrypty wstawiające przykładowe wiersze do tabel

```sql
INSERT INTO Customers(FirstName, LastName, PostalCode)
VALUES
('Zdzislaw', 'Wrona', '30-613'),
('Julia', 'Kowalczyk', '30-573'),
('Zofia', 'Nowak', '30-333'),
('Pawel', 'Wojcik', '30-815'),
('Marcin', 'Kowal', '30-715'),
('Aleksander', 'Wisniewski', '30-255'),
('Jakub', 'Nowak ', '30-355'),
('Julia', 'Wisniewska', '30-473'),
('Maja', 'Kaminska', '30-141'),
('Hanna', 'Zielinska', '30-564'),
('Mikolaj', 'Wozniak', '30-174'),
('Alicja', 'Lewandowska', '30-987'),
('Zuzanna', 'Szymanska', '30-574'),
('Leon', 'Adamczyk', '30-324'),
('Szymon', 'Wozniak', '30-174'),
('Lena', 'Wojcik', '30-574'),
('Filip', 'Wisniewski', '30-674'),
('Julia', 'Zielinski', '30-431')
```

```sql
INSERT INTO Employees(FirstName, LastName)
VALUES
('Agnieszka', 'Wrona'),
('Marcin', 'Kaczmarek'),
('Zofia', 'Nowak'),
('Lukasz', 'Wojciechowski'),
('Karolina', 'Krawczyk'),
('Mateusz', 'Dabrowski')
```

```sql
INSERT INTO ScreeningRooms(RoomID, RowsNumber)
VALUES
('A', 9),
('B', 9),
('C', 14),
('D', 10),
('E', 14)
```

```sql
INSERT INTO Movies (MovieTitle, ReleaseYear, RunningTime, Genre)
VALUES
('Bohemian Rhapsody', 2018, '2:13:00', 'Musical'),
('Wonder Woman', 2017, '2:29:00', 'Action'),
('Parasite', 2019, '2:12:00', 'Thriller'),
('Star Wars: The Rise of Skywalker', 2019, '2:22:00', 'Science Fiction'),
('Joker', 2019, '2:02:00', 'Thriller'),
('Raya and the Last Dragon', 2021, '1:57:00', 'Action'),
('Black Widow', 2021, '2:13:00', 'Superhero'),
('Jungle Cruise', 2021, '1:30:00', 'Adventure'),
('Free Guy', 2021, '2:15:00', 'Comedy')
```

```sql
INSERT INTO TicketPrices(TicketType, Price, EndDate)
VALUES
('monday', 15, '2021-01-13'),
('monday', 16, NULL),
('wednesday', 14.90 , '2021-01-13'),
('wednesday', 15.90 , NULL),
('regular', 18, '2021-01-13'),
('regular', 20, NULL)
```

```sql
INSERT INTO Screenings (MovieID, [Date], RoomID) VALUES
(5, '2021-02-18 13:00:00', 'E'),
(6, '2021-02-18 15:00:00', 'B'),
(2, '2021-02-18 16:00:00', 'C'),
(1, '2021-02-18 19:00:00', 'C'),
(6, '2021-02-18 20:00:00', 'E'),
(4, '2021-02-19 19:00:00', 'B'),
(3, '2021-02-19 20:00:00', 'C'),
(0, '2021-02-24 13:00:00', 'A'),
(0, '2021-02-28 17:00:00', 'B'),
(8, '2021-03-01 18:00:00', 'B'),
(7, '2021-03-01 20:00:00', 'C'),
(4, '2021-03-01 14:00:00', 'E'),
(3, '2021-03-01 14:00:00', 'D'),
(6, '2021-03-01 14:00:00', 'C'),
(5, '2021-03-02 13:00:00', 'E'),
(6, '2021-03-02 15:00:00', 'B'),
(2, '2021-03-02 16:00:00', 'C'),
```

```
(1, '2021-03-02 19:00:00', 'C'),
(6, '2021-03-02 20:00:00', 'E')
```

```
GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (100), (101)
EXEC make_reservation @SeatsTab, 1, 3, default

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (7), (8), (9)
EXEC make_reservation @SeatsTab, 1, 2, default

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (50), (51)
EXEC make_reservation @SeatsTab, 1, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (40), (41), (42), (43)
EXEC make_reservation @SeatsTab, 1, default, 6

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (30), (31)
EXEC make_reservation @SeatsTab, 1, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (90), (91), (92), (93), (94)
EXEC make_reservation @SeatsTab, 1, default, 4

```

```
GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (100), (101)
EXEC make_reservation @SeatsTab, 5, 3, default

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (7), (8), (9)
EXEC make_reservation @SeatsTab, 5, 2, default

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (50), (51)
EXEC make_reservation @SeatsTab, 5, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (40), (41), (42), (43)
EXEC make_reservation @SeatsTab, 5, default, 6

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (30), (31)
EXEC make_reservation @SeatsTab, 5, default, 4
GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (90), (91), (92), (93), (94)
EXEC make_reservation @SeatsTab, 5, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (100), (101)
```

```
EXEC make_reservation @SeatsTab, 12, 3, default

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (7), (8), (9)
EXEC make_reservation @SeatsTab, 12, 2, default

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (50), (51)
EXEC make_reservation @SeatsTab, 12, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (40), (41), (42), (43)

EXEC make_reservation @SeatsTab, 12, default, 6

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (30), (31)
EXEC make_reservation @SeatsTab, 12, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (80), (81), (82), (83), (84)

EXEC make_reservation @SeatsTab, 12, default, 4

GO
DECLARE @SeatsTab SeatsTableType
INSERT INTO @SeatsTab VALUES
  (100), (101)
EXEC make_reservation @SeatsTab, 12, 3, default
```

```
115
116 GO
117 DECLARE @SeatsTab SeatsTableType
118 INSERT INTO @SeatsTab VALUES
119   (7), (8), (9)
120 EXEC make_reservation @SeatsTab, 12, 2, default
121
122 GO
123 DECLARE @SeatsTab SeatsTableType
124 INSERT INTO @SeatsTab VALUES
125   (50), (51)
126 EXEC make_reservation @SeatsTab, 12, default, 4
127
128 GO
129 DECLARE @SeatsTab SeatsTableType
130 INSERT INTO @SeatsTab VALUES
131   (40), (41), (42), (43)
132 EXEC make_reservation @SeatsTab, 12, default, 6
133
134 GO
135 DECLARE @SeatsTab SeatsTableType
136 INSERT INTO @SeatsTab VALUES
137   (30), (31)
138 EXEC make_reservation @SeatsTab, 12, default, 4
139
140 GO
141 DECLARE @SeatsTab SeatsTableType
142 INSERT INTO @SeatsTab VALUES
143   (80), (81), (82), (83), (84)
144 EXEC make_reservation @SeatsTab, 12, default, 4
145
146 GO
147 DECLARE @SeatsTab SeatsTableType
148 INSERT INTO @SeatsTab VALUES
149   (70), (71), (72), (73), (74)
150 EXEC make_reservation @SeatsTab, 12, default, 3
151
152 GO
153 DECLARE @SeatsTab SeatsTableType
```

```
154 INSERT INTO @SeatsTab VALUES
155   (120), (121)
156 EXEC make_reservation @SeatsTab, 12, default, 3
157
158 GO
159 DECLARE @SeatsTab SeatsTableType
160 INSERT INTO @SeatsTab VALUES
161   (14), (15), (16)
162 EXEC make_reservation @SeatsTab, 12, 5, default
163
164 GO
165 DECLARE @SeatsTab SeatsTableType
166 INSERT INTO @SeatsTab VALUES
167   (35), (36)
168
169 EXEC make_reservation @SeatsTab, 1, 3, default
```