МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра систем автоматизированного проектирования (САПР)

отчет

по индивидуальному домашнему заданию

по дисциплине «Базы данных»

Тема: **«Разработка БД для обеспечения продажи билетов в кинотеатр (Вариант 32)»**

Студент гр. 3311 _____ Баймухамедов Р. Р.

Преподаватель _____ Новакова Н. Е.

Санкт-Петербург

2025

ЗАДАНИЕ

НА ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

Студент Баймухамедов Р. Р.

Группа 3311

Тема работы: «Разработка БД для обеспечения продажи билетов в кинотеатр (Вариант 32)»

Дата выдачи задания: 23.10.2025

Дата сдачи работы: 00.12.2025

Дата защиты работы: 00.12.2025

Студент _____ Баймухамедов Р. Р.

Преподаватель _____ Новакова Н. Е.

1. Аннотация

В рамках данной работы разработана база данных CinemaDB, предназначенная для хранения информации о фильмах, кинозалах, сеансах, местах и операциях по бронированию и продаже билетов. Спроектированы таблицы с корректными связями и ограничениями целостности данных, реализованы хранимые процедуры для добавления основных сущностей и выполнения ключевых операций (бронирование, продажа, отмена брони). Для получения аналитической информации (заполняемость залов, выручка по сеансам) созданы пользовательские функции и представления, упрощающие доступ к данным. Дополнительно разработаны триггеры, предотвращающие двойное бронирование мест и обеспечивающие журналирование изменений статуса билетов. Сформирована стратегия резервного копирования, а также подробно описан процесс восстановления базы, что обеспечивает сохранность и восстановимость информации. Разработанная база данных готова к использованию в системе управления кинотеатром.

Summary

This work presents the design and implementation of the CinemaDB database intended to store and manage information about movies, cinema halls, screenings, seats, and ticket reservations and sales. The schema includes tables with proper relationships and data integrity constraints; stored procedures were implemented for inserting core entities and performing key operations (ticket reservation, sale, and cancellation). User-defined functions and views were developed to support analytical queries such as hall occupancy, screening revenue, and customer activity. Additionally, triggers were created to prevent double booking of seats and to log changes of ticket status in an audit table. A backup strategy and a step-by-step recovery process were defined to ensure data safety and recoverability. The resulting database is ready to be used as a backend for a cinema management and ticketing system.

Оглавление

1. Аннотация 3
2. Цель работы 5
3. Формулировка задания 6
4. Формулировка задачи 7
5. Проектирование БД 8
6. Создание базы данных в Microsoft SQL Server 10
7. Диаграмма связи таблицы базы данных 11
8. Описание структуры таблиц БД 12
9. Создание схемы для таблиц 21
10. Создание таблиц в Microsoft SQL Server 22
11. Заполнение таблиц данными 29
12. Разработка объектов промежуточного слоя 35
13. Триггеры 67
14. Стратегия резервного копирования 73
15. Создание резервного копирования 75
16. Восстановление базы данных 78
17. Заключение 81
18. Список использованных источников 83

2. Цель работы

Закрепить теоретические знания, полученные на дисциплине “Базы Данных”, а также для получения практических навыков в проектировании и создании базы данных. Необходимо создать объекты промежуточного слоя, триггеры и определить стратегию резервного копирования

3. Формулировка задания

Разработка комплексной реляционной базы данных “CinemaDB” для автоматизации продажи билетов в кинотеатре: ведение каталога фильмов, залов и мест, расписания сеансов, учёта клиентов, бронирований, оплат и билетов. Обеспечение целостности данных (PK/FK, CHECK), реализация логики бронирования и покупки при помощи транзакций.

4. Формулировка задачи

Спроектировать и реализовать информационную систему для кинотеатра, позволяющую:

- Вести учёт фильмов с их характеристиками (название, оригинальное название, длительность, возрастной рейтинг, краткое описание)
- Хранить информацию о залах кинотеатра и их конфигурации (тип зала, ряды и места)
- Формировать расписание сеансов с указанием фильма, зала, времени начала и базовой стоимости билета
- Регистрировать покупателей с их контактными данными (ФИО, e-mail, телефон, дата регистрации)
- Осуществлять бронирование и продажу билетов на конкретные места в зале на выбранный сеанс с фиксацией статуса (бронь, продан, отменён) и цены
- Определять занятость мест на сеанс (свободно, забронировано, продано) на основе данных о проданных и забронированных билетах
- Обеспечивать надёжное хранение данных, включая систему резервного копирования и восстановления базы
- Предоставлять средства для анализа работы кинотеатра (загрузка залов, выручка по фильмам и сеансам, активность покупателей) с использованием представлений, функций и хранимых процедур

5. Проектирование БД

Описания связей:

- Один покупатель может иметь несколько билетов.
- Один фильм может соответствовать многим сеансам.
- Один зал содержит множество мест.
- Один сеанс проходит в одном зале и относится к одному фильму, но может включать множество билетов (на разные места).
- Одно место в зале может участвовать во множестве сеансов (через разные записи о билетах).

6. Создание базы данных в Microsoft SQL Server

Создадим базу данных CinemaDB с заданными парамметрами:

- Имя базы данных
- Каталог хранения файлов данных и журнала
- Начальный размер файла данных и файла журнала с автоувеличением и без верхнего предела

Запрос

```
CREATE DATABASE CinemaDB
ON PRIMARY
(
   NAME = N'CinemaDB',
   FILENAME = N'C:\GitHub\course-3\db\idz\CinemaDB.mdf',
   SIZE = 8MB,
   FILEGROWTH = 16MB
)
LOG ON
(
   NAME = N'CinemaDB_log',
   FILENAME = N'C:\GitHub\course-3\db\idz\CinemaDB_log.ldf',
   SIZE = 8MB,
   FILEGROWTH = 16MB
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:24:48.9506575+03:00

7. Диаграмма связи таблицы базы данных

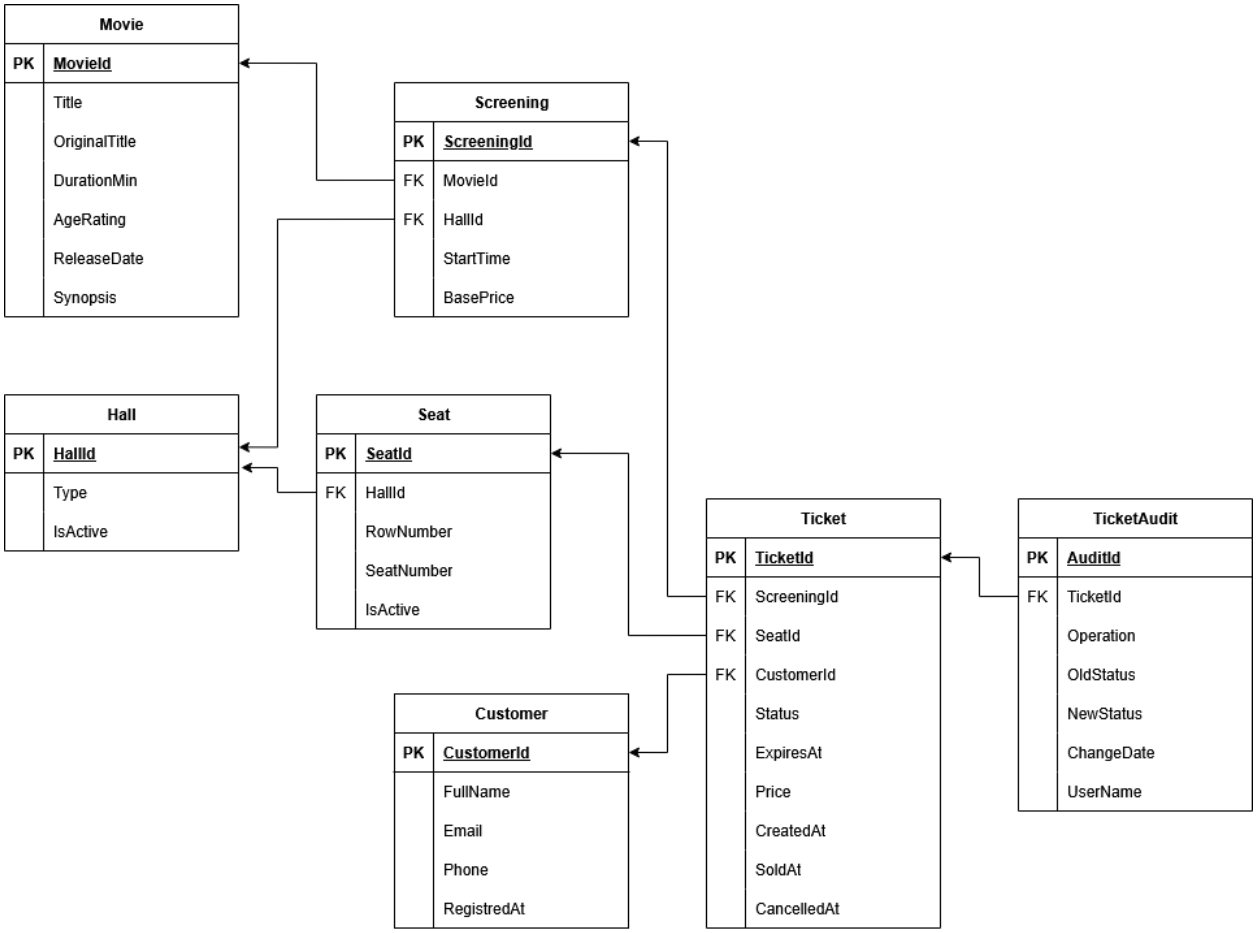

Рисунок 1

8. Описание структуры таблиц БД

Customer (Покупатель)

Описание структуры таблицы БД	Наименование таблицы БД: Покупатель	Имя таблицы: Customer
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 1		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	CustomerId	INT IDENTITY(1,1)	P	NOT NULL
2	ФИО	FullName	NVARCHAR(255)		NOT NULL
3	Email	Email	NVARCHAR(255)	U	NOT NULL; уникальность Email
4	Телефон	Phone	NVARCHAR(32)	U	NOT NULL; уникальность Phone
5	Дата регистрации	RegistredAt	DATETIME2(0)		NOT NULL; DEFAULT = SYSUTCDATETIME()

Movie (Фильм)

Описание структуры таблицы БД	Наименование таблицы БД: Фильм	Имя таблицы: Movie
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 2		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	MovieId	INT IDENTITY(1,1)	P	NOT NULL
2	Название	Title	NVARCHAR(255)		NOT NULL
3	Оригинальное название	OriginalTitle	NVARCHAR(255)		NULL
4	Длительность	DurationMin	SMALLINT		NOT NULL; [DurationMin] BETWEEN 1 AND 1000
5	Возрастной рейтинг	AgeRating	NVARCHAR(4)		NOT NULL; [AgeRating] IN (N'0+',N'6+',N'12+',N'16+',N'18+')
6	Дата релиза	ReleaseDate	DATE		NULL

7	Описание	Synopsis	NVARCHAR(2000)		NULL

Hall (Зал)

Описание структуры таблицы БД	Наименование таблицы БД: Зал	Имя таблицы: Hall
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 3		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	HallId	SMALLINT IDENTITY(1,1)	P	
2	Тип зала	Type	NVARCHAR(20)		NOT NULL; [Type] IN (N'STANDARD', N'3D', N'IMAX', N'VIP')
3	Активен	IsActive	BIT		NOT NULL; DEFAULT = 1

Seat (Место)

Описание структуры таблицы БД	Наименование таблицы БД: Место	Имя таблицы: Seat
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 4		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	SeatId	INT IDENTITY(1,1)	P	
2	Зал	HallId	INT	FK	NOT NULL; REFERENCES Hall(HallId)
3	Ряд	RowNumber	SMALLINT		NOT NULL; [RowNumber] >= 1
4	Место	SeatNumber	DATETIME2(0)		NOT NULL; [SeatNumber] >= 1
5	Активно	IsActive	DECIMAL(10, 2)		NOT NULL; DEFAULT = 1

Screening (Сеанс)

Описание структуры таблицы БД	Наименование таблицы БД: Сеанс	Имя таблицы: Screening
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 5		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	ScreeningId	INT IDENTITY(1,1)	P	
2	Фильм	MovieId	INT	FK	NOT NULL; REFERENCES Movie(MovieId)
3	Зал	HallId	SMALLINT	FK	NOT NULL; REFERENCES Hall(HallId)
4	Начало сеанса	StartTime	DATETIME2(0)		NOT NULL
5	Базовая цена	BasePrice	DECIMAL(10, 2)		NOT NULL; [BasePrice] >= 0

Ticket (Билет)

Описание структуры таблицы БД	Наименование таблицы БД: Билет	Имя таблицы: Ticket
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 6		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	TicketId	INT IDENTITY(1,1)	P	
2	Сеанс	ScreeningId	INT	FK	NOT NULL; REFERENCES Screening(ScreeningId)
3	Место	SeatId	INT	FK	NOT NULL; REFERENCES Seat(SeatId)
4	Покупатель	CustomerId	INT	FK	NOT NULL; REFERENCES Customer(CustomerId)

5	Статус	Status	NVARCHAR(10)		NOT NULL; [Status] IN (N'Reserved', N'Sold', N'Cancelled')
6	Истекает (для брони)	ExpiresAt	DATETIME2(0)		NULL
7	Цена	Price	DECIMAL(10, 2)		NOT NULL; [Price] >= 0
8	Создано	CreatedAt	DATETIME2(0)		NOT NULL; DEFAULT = SYSUTCDATETIME()
9	Продано	SoldAt	DATETIME2(0)		NULL
10	Отменено	CancelledAt	DATETIME2(0)		NULL

TicketAudit (Журнал билетов)

Описание структуры таблицы БД	Наименование таблицы БД: Журнал билетов	Имя таблицы: TicketAudit
Дата разработки: 19.11.2025		
Порядковый номер таблицы: 7		

№ п/п	Наименование поля	Спецификация данных			
		Имя поля	Тип данных	Ключ	Ограничения целостности
1	Идентификатор	AuditId	INT IDENTITY(1,1)	P	
2	Билет	TicketId	INT	FK	NOT NULL; REFERENCES Ticket(TicketId)
3	Тип операции	Operation	NVARCHAR(10)		NOT NULL; IN (N’INSERT’, N’UPDATE', N’DELETE')
4	Старый статус билета	OldStatus	NVARCHAR(10)		NULL; IN (NULL, N’INSERT’, N’UPDATE', N’DELTE')

5	Новый статус билета	NewStatus	NVARCHAR(10)		NULL; IN (NULL, N’INSERT’, N’UPDATE', N’DELTE')
6	Дата и время изменения	ChangeDate	DATETIME2(0)		NOT NULL; DEFAULT = SYSUTCDATETIME()
7	Имя пользователя	UserName	NVARCHAR(128)		NOT NULL; [Price] >= 0

9. **Создание схемы для таблиц**

Запрос

CREATE SCHEMA idz

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-04T17:51:27.6433744+03:00

10.Создание таблиц в Microsoft SQL Server

Создадим таблицу Customer

Запрос

```
USE CinemaDB;
GO

CREATE TABLE Customer (
   CustomerId INT IDENTITY(1,1) NOT NULL
      CONSTRAINT PK_Customer PRIMARY KEY,
   FullName NVARCHAR(255) NOT NULL,
   Email NVARCHAR(255) NOT NULL,
   Phone NVARCHAR(32) NOT NULL,
   RegisteredAt DATETIME2(0) NOT NULL
      CONSTRAINT DF_Customer_RegisteredAt DEFAULT
(SYSUTCDATETIME()),
   CONSTRAINT UQ_Customer_Email UNIQUE (Email),
   CONSTRAINT UQ_Customer_Phone UNIQUE (Phone)
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:54:04.3057672+03:00

Создадим таблицу Movie

Запрос

```
USE CinemaDB;
GO

CREATE TABLE Movie (
   MovieId INT IDENTITY(1,1) NOT NULL
      CONSTRAINT PK_Movie PRIMARY KEY,
```

```
   Title NVARCHAR(255) NOT NULL,
   OriginalTitle NVARCHAR(255) NULL,
   DurationMin SMALLINT NOT NULL
      CONSTRAINT CK_Movie_DurationMin CHECK (DurationMin BETWEEN
1 AND 1000),
   AgeRating NVARCHAR(4) NOT NULL
      CONSTRAINT CK_Movie_AgeRating CHECK (AgeRating IN
(N'0+',N'6+',N'12+',N'16+',N'18+')),
   ReleaseDate DATE NULL,
   Synopsis NVARCHAR(2000) NULL,
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:52:15.8188457+03:00

Создадим таблицу Hall

Запрос

```
USE CinemaDB;
GO

CREATE TABLE Hall (
   HallId SMALLINT IDENTITY(1,1) NOT NULL CONSTRAINT PK_Hall
PRIMARY KEY,
   Type NVARCHAR(20) NOT NULL,
   IsActive BIT NOT NULL CONSTRAINT DF_Hall_IsActive DEFAULT (1),
   CONSTRAINT CK_Hall_Type CHECK (Type IN (N'STANDARD', N'3D',
N'IMAX', N'VIP'))
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:53:28.7701176+03:00

Создадим таблицу Seat

Запрос

```
USE CinemaDB;
GO

CREATE TABLE Seat (
   SeatId INT IDENTITY(1,1) NOT NULL CONSTRAINT PK_Seat PRIMARY KEY,
   HallId SMALLINT NOT NULL,
   RowNumber INT NOT NULL CHECK (RowNumber >= 1),
   SeatNumber INT NOT NULL CHECK (SeatNumber >= 1),
   IsActive BIT NOT NULL CONSTRAINT DF_Seat_IsActive DEFAULT (1),
   CONSTRAINT FK_Seat_Hall FOREIGN KEY (HallId) REFERENCES Hall(HallId),
   CONSTRAINT UQ_Seat_Coord UNIQUE (HallId, RowNumber, SeatNumber)
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:54:44.4437292+03:00

Создадим таблицу Screening

Запрос

```
USE CinemaDB;
GO

CREATE TABLE Screening (
   ScreeningId INT IDENTITY(1,1) NOT NULL CONSTRAINT PK_Screening PRIMARY KEY,
```

```
   MovieId INT NOT NULL,
   HallId SMALLINT NOT NULL,
   StartTime DATETIME2(0) NOT NULL,
   BasePrice DECIMAL(10,2) NOT NULL CHECK (BasePrice >= 0),
   CONSTRAINT FK_Screening_Movie FOREIGN KEY (MovieId) REFERENCES Movie(MovieId),
   CONSTRAINT FK_Screening_Hall FOREIGN KEY (HallId) REFERENCES Hall(HallId)
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:55:04.1476246+03:00

Создадим таблицу Ticket

Запрос

```
USE CinemaDB;
GO

CREATE TABLE Ticket (
   TicketId INT IDENTITY(1,1) NOT NULL CONSTRAINT PK_Ticket PRIMARY KEY,
   ScreeningId INT NOT NULL,
   SeatId INT NOT NULL,
   CustomerId INT NOT NULL,
   Status NVARCHAR(10) NOT NULL CONSTRAINT CK_Ticket_Status CHECK (Status IN (N'Reserved', N'Sold', N'Cancelled')),
   ExpiresAt DATETIME2(0) NULL,
   Price DECIMAL(10,2) NOT NULL CHECK (Price >= 0),
   CreatedAt DATETIME2(0) NOT NULL CONSTRAINT DF_Ticket_CreatedAt DEFAULT (SYSUTCDATETIME()),
```

```
   SoldAt DATETIME2(0) NULL,
   CancelledAt DATETIME2(0) NULL,
   CONSTRAINT FK_Ticket_Screening FOREIGN KEY (ScreeningId) REFERENCES Screening(ScreeningId),
   CONSTRAINT FK_Ticket_Seat FOREIGN KEY (SeatId) REFERENCES Seat(SeatId)
   CONSTRAINT FK_Ticket_Customer FOREIGN KEY (CustomerId) REFERENCES Customer(CustomerId)
);
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-20T11:57:35.8060897+03:00

Создадим таблицу TicketAudit

Запрос

```
USE CinemaDB;
GO

CREATE TABLE TicketAudit (
   AuditId INT IDENTITY(1,1) NOT NULL
      CONSTRAINT PK_TicketAudit PRIMARY KEY,
   TicketId INT NOT NULL,
   Operation NVARCHAR(10) NOT NULL
      CONSTRAINT CK_TicketAudit_Operation
      CHECK (Operation IN (N'INSERT', N'UPDATE', N'DELETE')),
   OldStatus NVARCHAR(10) NULL,
      CONSTRAINT CK_TicketAudit_OldStatus
      CHECK (OldStatus IS NULL OR OldStatus IN (N'Reserved', N'Sold', N'Cancelled')),
   NewStatus NVARCHAR(10) NULL,
```

```
    CONSTRAINT CK_TicketAudit_NewStatus
    CHECK (NewStatus IS NULL OR NewStatus IN (N'Reserved', N'Sold',
N'Cancelled')),
  ChangeDate DATETIME2(0) NOT NULL
    CONSTRAINT DF_TicketAudit_ChangeDate
    DEFAULT (SYSUTCDATETIME()),
  UserName NVARCHAR(128) NULL,
  CONSTRAINT FK_TicketAudit_Ticket
  FOREIGN KEY (TicketId) REFERENCES Ticket(TicketId)
);
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-03T19:05:56.8966752+03:00

11.Заполнение таблиц данными

Заполним таблицу Customer

Запрос

```
USE CinemaDB;
GO

INSERT INTO Customer (FullName, Email, Phone)
VALUES
(N'Иванов Иван Иванович', N'ivanov@example.com', N'+79001112233'),
(N'Петров Пётр Петрович', N'petrov@example.com', N'+79002223344'),
(N'Сидорова Анна Сергеевна', N'sidorova@example.com', N'+79003334455');
GO
```

Результат

(Затронуто строк: 3)

Время выполнения: 2025-11-20T12:22:48.1045595+03:00

Заполним таблицу Movie

Запрос

```
USE CinemaDB;
GO

INSERT INTO Movie (Title, OriginalTitle, DurationMin, AgeRating, ReleaseDate,
Synopsis)
VALUES
(N'Интерстеллар', N'Interstellar', 169, N'12+', '2014-11-06',
 N'Группа исследователей отправляется сквозь червоточину в космосе в
поисках нового дома для человечества.'),
(N'Начало', N'Inception', 148, N'16+', '2010-07-22', NULL),
(N'Титаник', N'Titanic', 195, N'12+', '1997-12-19',
 N'История любви на борту легендарного океанского лайнера.');
```

```
GO
```

Результат

(Затронуто строк: 3)

Время выполнения: 2025-11-20T12:33:36.0743481+03:00

Заполним таблицу Hall

Запрос

```
USE CinemaDB;
GO

INSERT INTO Hall (Type)
VALUES
(N'STANDARD'),
(N'3D'),
(N'IMAX');
GO
```

Результат

(Затронуто строк: 3)

Время выполнения: 2025-11-20T12:39:21.5177879+03:00

Заполним таблицу Seat. Создадим места для трех залов.

Запрос

```
USE CinemaDB;
GO

DECLARE @row INT, @seat INT;

SET @row = 1;
WHILE @row <= 5
BEGIN
```

```
    SET @seat = 1;
    WHILE @seat <= 10
    BEGIN
        INSERT INTO Seat (HallId, RowNumber, SeatNumber)
        VALUES (1, @row, @seat);

        SET @seat = @seat + 1;
    END;

    SET @row = @row + 1;
END;
GO
```

Результат

(Затронута 1 строка)

(Затронута 1 строка)

…

(Затронута 1 строка)

(Затронута 1 строка)

Время выполнения: 2025-11-20T12:42:28.0616696+03:00

Запрос

```
USE CinemaDB;
GO

DECLARE @row INT, @seat INT;

SET @row = 1;
WHILE @row <= 3
BEGIN
    SET @seat = 1;
```

```
    WHILE @seat <= 8
    BEGIN
        INSERT INTO Seat (HallId, RowNumber, SeatNumber)
        VALUES (2, @row, @seat);

        SET @seat = @seat + 1;
    END;

    SET @row = @row + 1;
END;
GO
```

Результат

(Затронута 1 строка)

(Затронута 1 строка)

…

(Затронута 1 строка)

(Затронута 1 строка)

Время выполнения: 2025-11-20T12:45:20.0626473+03:00

Запрос

```
USE CinemaDB;
GO

DECLARE @row INT, @seat INT;

SET @row = 1;
WHILE @row <= 3
BEGIN
    SET @seat = 1;
    WHILE @seat <= 6
```

```
    BEGIN
        INSERT INTO Seat (HallId, RowNumber, SeatNumber)
        VALUES (3, @row, @seat);

        SET @seat = @seat + 1;
    END;

    SET @row = @row + 1;
END;
GO
```

Результат

(Затронута 1 строка)

(Затронута 1 строка)

…

(Затронута 1 строка)

(Затронута 1 строка)

Время выполнения: 2025-11-20T12:46:30.2788205+03:00

Заполним таблицу Screening

Запрос

```
USE CinemaDB;
GO

INSERT INTO Screening (MovieId, HallId, StartTime, BasePrice)
VALUES
(1, 1, '2025-11-21 18:00:00', 350.00),  -- Интерстеллар, зал 1
(2, 1, '2025-11-21 21:00:00', 350.00),  -- Начало, зал 1
(3, 2, '2025-11-22 19:00:00', 400.00);  -- Титаник, зал 2
GO
```

Результат

(Затронуто строк: 3)

Время выполнения: 2025-11-20T12:47:58.1154740+03:00

Заполним таблицу Ticket

Запрос

```
USE CinemaDB;
GO

INSERT INTO Ticket (ScreeningId, SeatId, CustomerId, Status, ExpiresAt, Price)
VALUES
(1, 1, 1, N'Sold', NULL, 350.00),
(1, 2, 2, N'Sold', NULL, 350.00),
(1, 3, 3, N'Reserved', DATEADD(MINUTE, 15, SYSUTCDATETIME()), 350.00),
(2, 10, 1, N'Sold', NULL, 350.00),
(3, 25, 2, N'Reserved', DATEADD(MINUTE,Результат
```

(Затронуто строк: 5)

Время выполнения: 2025-11-20T12:53:53.4533449+03:00

12.Разработка объектов промежуточного слоя

Хранимые процедуры

Создадим хранимую процедуру AddCustomer, которая добавляет нового покупателя с проверкой уникальности e-mail и телефона и возвращает его идентификатор.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE AddCustomer
  @FullName   NVARCHAR(255),
  @Email      NVARCHAR(255),
  @Phone      NVARCHAR(32),
  @CustomerId INT OUTPUT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    IF EXISTS (SELECT 1 FROM Customer WHERE Email = @Email)
    BEGIN
      RAISERROR(N'Покупатель с таким Email уже существует.', 16, 1);
      RETURN;
    END;

    IF EXISTS (SELECT 1 FROM Customer WHERE Phone = @Phone)
    BEGIN
      RAISERROR(N'Покупатель с таким номером телефона уже существует.',
16, 1);
      RETURN;
```

```
    END;

    INSERT INTO Customer (FullName, Email, Phone)
    VALUES (@FullName, @Email, @Phone);

    SET @CustomerId = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT N'Покупатель успешно добавлен.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:36:00.7714498+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @CustomerId INT;

EXEC AddCustomer
  @FullName = N'Кузнецов Алексей Сергеевич',
  @Email = N'akuznetsov@example.com',
```

```
  @Phone = N'+79004445566',
  @CustomerId = @CustomerId OUTPUT;

SELECT @CustomerId AS CustomerId;
GO
```

Результат

CustomerId

4

Создадим хранимую процедуру AddMovie, которая добавляет новый фильм с указанными характеристиками и возвращает его идентификатор.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE AddMovie
  @Title NVARCHAR(255),
  @OriginalTitle NVARCHAR(255) = NULL,
  @DurationMin SMALLINT,
  @AgeRating NVARCHAR(4),
  @ReleaseDate DATE = NULL,
  @Synopsis NVARCHAR(2000) = NULL,
  @MovieId INT OUTPUT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    INSERT INTO Movie (Title, OriginalTitle, DurationMin, AgeRating,
ReleaseDate, Synopsis)
```

```
    VALUES (@Title, @OriginalTitle, @DurationMin, @AgeRating, @ReleaseDate, @Synopsis);

    SET @MovieId = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT N'Фильм успешно добавлен.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:36:28.4083707+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @MovieId INT;

EXEC AddMovie
  @Title = N'Матрица',
  @OriginalTitle = N'The Matrix',
  @DurationMin = 136,
  @AgeRating = N'16+',
```

```
  @ReleaseDate = '1999-03-31',
  @Synopsis = N'Хакер Нео узнаёт, что реальность - это компьютерная симуляция.',
  @MovieId = @MovieId OUTPUT;

SELECT @MovieId AS MovieId;
GO
```

Результат

MovieId

4

Создадим хранимую процедуру AddHall, которая создаёт новый зал с заданным типом и признаком активности и возвращает его идентификатор.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE AddHall
  @Type NVARCHAR(20),
  @IsActive BIT = 1,
  @HallId SMALLINT OUTPUT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    INSERT INTO Hall (Type, IsActive)
    VALUES (@Type, @IsActive);

    SET @HallId = SCOPE_IDENTITY();
```

```
    COMMIT TRANSACTION;
    PRINT N'Зал успешно добавлен.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:38:18.4888795+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @HallId SMALLINT;

EXEC AddHall
  @Type = N'IMAX',
  @IsActive = 1,
  @HallId = @HallId OUTPUT;

SELECT @HallId AS HallId;
GO
```

Результат

HallId

4

Создадим хранимую процедуру AddSeat, которая добавляет новое место в указанном зале с проверкой существования зала и отсутствия дубликата координат, и возвращает идентификатор места.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE AddSeat
  @HallId     SMALLINT,
  @RowNumber  INT,
  @SeatNumber INT,
  @IsActive   BIT = 1,
  @SeatId     INT OUTPUT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    IF NOT EXISTS (SELECT 1 FROM Hall WHERE HallId = @HallId AND
IsActive = 1)
    BEGIN
      RAISERROR(N'Указанный зал не найден или отключён.', 16, 1);
      RETURN;
    END;

    IF EXISTS (
        SELECT 1
        FROM Seat
```

```
        WHERE HallId = @HallId
          AND RowNumber = @RowNumber
          AND SeatNumber = @SeatNumber
    )
    BEGIN
      RAISERROR(N'Место с такими координатами в этом зале уже существует.', 16, 1);
      RETURN;
    END;

    INSERT INTO Seat (HallId, RowNumber, SeatNumber, IsActive)
    VALUES (@HallId, @RowNumber, @SeatNumber, @IsActive);

    SET @SeatId = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT N'Место успешно добавлено.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:40:07.2176837+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @SeatId INT;
DECLARE @HallId SMALLINT;
SET @HallId = 4;

EXEC AddSeat
  @HallId = @HallId,
  @RowNumber = 1,
  @SeatNumber = 1,
  @IsActive = 1,
  @SeatId = @SeatId OUTPUT;

SELECT @SeatId AS SeatId;
GO
```

Результат

SeatId

93

Создадим хранимую процедуру AddScreening, которая добавляет новый сеанс для выбранного фильма в заданном зале, проверяет корректность и возвращает идентификатор сеанса.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE AddScreening
  @MovieId     INT,
  @HallId      SMALLINT,
```

```
  @StartTime   DATETIME2(0),
  @BasePrice   DECIMAL(10,2),
  @ScreeningId INT OUTPUT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    IF NOT EXISTS (SELECT 1 FROM Movie WHERE MovieId = @MovieId)
    BEGIN
      RAISERROR(N'Фильм с указанным MovieId не найден.', 16, 1);
      RETURN;
    END;

    IF NOT EXISTS (SELECT 1 FROM Hall WHERE HallId = @HallId AND
IsActive = 1)
    BEGIN
      RAISERROR(N'Зал с указанным HallId не найден или отключён.', 16, 1);
      RETURN;
    END;

    INSERT INTO Screening (MovieId, HallId, StartTime, BasePrice)
    VALUES (@MovieId, @HallId, @StartTime, @BasePrice);

    SET @ScreeningId = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT N'Сеанс успешно добавлен.';
  END TRY
  BEGIN CATCH
```

```
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:43:29.2719890+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @ScreeningId INT;
DECLARE @HallId SMALLINT;
SET @HallId = 4;
DECLARE @MovieId INT;
SET @MovieId = 4;

EXEC AddScreening
  @MovieId = @MovieId,
  @HallId = @HallId,
  @StartTime = '2025-12-01 19:00:00',
  @BasePrice = 500.00,
  @ScreeningId = @ScreeningId OUTPUT;

SELECT @ScreeningId AS ScreeningId;
GO
```

Результат

ScreeningId

4

Создадим хранимую процедуру ReserveTicket, которая оформляет бронь билета для указанного покупателя, сеанса и места, проверяет доступность места и возвращает идентификатор созданного билета.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE ReserveTicket
  @CustomerId  INT,
  @ScreeningId INT,
  @SeatId      INT,
  @HoldMinutes INT = 15,
  @TicketId    INT OUTPUT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @now   DATETIME2(0) = SYSUTCDATETIME();
    DECLARE @price DECIMAL(10,2);

    IF NOT EXISTS (SELECT 1 FROM Customer WHERE CustomerId =
@CustomerId)
    BEGIN
      RAISERROR(N'Покупатель с указанным CustomerId не найден.', 16, 1);
      RETURN;
    END;
```

```
  IF NOT EXISTS (SELECT 1 FROM Screening WHERE ScreeningId =
@ScreeningId)
  BEGIN
    RAISERROR(N'Сеанс с указанным ScreeningId не найден.', 16, 1);
    RETURN;
  END;

  IF NOT EXISTS (SELECT 1 FROM Seat WHERE SeatId = @SeatId AND
IsActive = 1)
  BEGIN
    RAISERROR(N'Место с указанным SeatId не найдено или отключено.', 16,
1);
    RETURN;
  END;

  IF NOT EXISTS (
    SELECT 1
    FROM Screening sc
    JOIN Seat se ON sc.HallId = se.HallId
    WHERE sc.ScreeningId = @ScreeningId
      AND se.SeatId = @SeatId
  )
  BEGIN
    RAISERROR(N'Указанное место не принадлежит залу данного сеанса.',
16, 1);
    RETURN;
  END;

  IF EXISTS (
```

```
    SELECT 1
    FROM Ticket t
    WHERE t.ScreeningId = @ScreeningId
      AND t.SeatId      = @SeatId
      AND (
          t.Status = N'Sold'
          OR (t.Status = N'Reserved' AND t.ExpiresAt > @now)
        )
  )
  BEGIN
    RAISERROR(N'Место уже занято (продано или находится в активной
броне).', 16, 1);
    RETURN;
  END;

  SELECT @price = BasePrice
  FROM Screening
  WHERE ScreeningId = @ScreeningId;

  INSERT INTO Ticket (ScreeningId, SeatId, CustomerId, Status, ExpiresAt,
Price)
  VALUES (
    @ScreeningId,
    @SeatId,
    @CustomerId,
    N'Reserved',
    DATEADD(MINUTE, @HoldMinutes, @now),
    @price
  );
```

```
    SET @TicketId = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT N'Бронь успешно оформлена.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:45:18.6813283+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @TicketId INT;
DECLARE @ScreeningId INT;
SET @ScreeningId = 4
DECLARE @SeatId INT;
SET @SeatId = 93;
DECLARE @CustomerId INT;
SET @CustomerId = 4;

EXEC ReserveTicket
  @CustomerId = @CustomerId,
```

```
  @ScreeningId = @ScreeningId,
  @SeatId = @SeatId,
  @HoldMinutes = 20, -- бронь на 20 минут
  @TicketId = @TicketId OUTPUT;

SELECT @TicketId AS TicketId_Reserved;
GO
```

Результат

TicketId_Reserved

6

Создадим хранимую процедуру SellTicket, которая переводит ранее забронированный и не истёкший билет в статус “Sold” и фиксирует время продажи.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE SellTicket
  @TicketId INT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @now DATETIME2(0) = SYSUTCDATETIME();

    UPDATE Ticket
    SET Status    = N'Sold',
        SoldAt    = @now,
```

```
      ExpiresAt = NULL
    WHERE TicketId = @TicketId
      AND Status = N'Reserved'
      AND (ExpiresAt IS NULL OR ExpiresAt > @now);

    IF @@ROWCOUNT = 0
    BEGIN
      RAISERROR(N'Невозможно продать билет: бронь не найдена, истекла или
билет уже не в статусе Reserved.', 16, 1);
      RETURN;
    END;

    COMMIT TRANSACTION;
    PRINT N'Билет успешно продан.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:49:18.6028889+03:00

Вызов процедуры

```
USE CinemaDB;
GO
```

```
DECLARE @TicketId INT;
SET @TicketId = 6

EXEC SellTicket
  @TicketId = @TicketId;

SELECT TicketId, ScreeningId, SeatId, Status, Price, CreatedAt, SoldAt
FROM Ticket
WHERE TicketId = @TicketId;
GO
```

Результат

Результаты | Сообщения

	TicketId	ScreeningId	SeatId	Status	Price	CreatedAt	SoldAt
1	6	4	93	Sold	500.00	2025-11-30 11:54:36	2025-11-30 11:57:00

Рисунок 2

(Затронута 1 строка)

Билет успешно продан.

(Затронута 1 строка)

Время выполнения: 2025-11-30T14:57:00.1710131+03:00

Создадим хранимую процедуру CancelReservation, которая отменяет действующую бронь билета, переводя его в статус “Cancelled” и обнуляя время истечения брони.

Запрос

```
USE CinemaDB;
GO

CREATE PROCEDURE CancelReservation
  @TicketId INT
AS
BEGIN
```

```
  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @now DATETIME2(0) = SYSUTCDATETIME();

    UPDATE Ticket
    SET Status      = N'Cancelled',
        CancelledAt = @now,
        ExpiresAt   = NULL
    WHERE TicketId = @TicketId
      AND Status = N'Reserved';

    IF @@ROWCOUNT = 0
    BEGIN
      RAISERROR(N'Невозможно отменить бронь: запись не найдена или уже
не в статусе Reserved.', 16, 1);
      RETURN;
    END;

    COMMIT TRANSACTION;
    PRINT N'Бронь успешно отменена.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

    PRINT N'Ошибка: ' + ERROR_MESSAGE();
  END CATCH
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T13:50:19.4912265+03:00

Вызов процедуры

```
USE CinemaDB;
GO

DECLARE @TicketId INT;
SET @TicketId = 6;

EXEC CancelReservation
  @TicketId = @TicketId;
```

Результат

(Затронуто строк: 0)

Ошибка: Невозможно отменить бронь: запись не найдена или уже не в статусе Reserved.

Время выполнения: 2025-11-30T17:59:08.6291835+03:00

Функции (UDF)

Создадим функцию GetAvailableSeatsCount, которая возвращает количество свободных (не проданных и не находящихся в активной броне) мест для указанного сеанса.

Запрос

```
USE CinemaDB;
GO

CREATE FUNCTION GetAvailableSeatsCount
(
  @ScreeningId INT
)
```

```
RETURNS INT
AS
BEGIN
  DECLARE @Now DATETIME2(0);
  DECLARE @Count INT;

  SET @Now = SYSUTCDATETIME();

  SELECT @Count = COUNT(*)
  FROM Screening sc
  JOIN Seat se
    ON se.HallId = sc.HallId
   AND se.IsActive = 1
  LEFT JOIN Ticket t
    ON t.ScreeningId = sc.ScreeningId
   AND t.SeatId = se.SeatId
   AND (
        t.Status = N'Sold'
        OR (t.Status = N'Reserved' AND t.ExpiresAt > @Now)
      )
  WHERE sc.ScreeningId = @ScreeningId
    AND t.TicketId IS NULL;

  RETURN @Count;
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T19:20:10.3466920+03:00

Вызов функции

```
SELECT GetAvailableSeatsCount(1) AS AvailableSeats;
```

Результат

AvailableSeats

48

Создадим функцию GetScreeningRevenue, которая вычисляет суммарную выручку по проданным билетам для указанного сеанса.

Запрос

```
USE CinemaDB;
GO

CREATE FUNCTION GetScreeningRevenue
(
  @ScreeningId INT
)
RETURNS DECIMAL(18,2)
AS
BEGIN
  DECLARE @Revenue DECIMAL(18,2);

  SELECT @Revenue = ISNULL(SUM(Price), 0.00)
  FROM Ticket
  WHERE ScreeningId = @ScreeningId
    AND Status = N'Sold';

  RETURN @Revenue;
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T19:33:34.1740778+03:00

Вызов функции

```
SELECT GetScreeningRevenue(1) AS ScreeningRevenue;
```

Результат

ScreeningRevenue

700.00

Создадим табличную функцию GetAvailableSeats, которая возвращает список свободных (не проданных и не находящихся в активной брони) мест для указанного сеанса.

Запрос

```
USE CinemaDB;
GO

CREATE FUNCTION GetAvailableSeats
(
    @ScreeningId INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        se.SeatId,
        se.RowNumber,
        se.SeatNumber
    FROM Screening sc
    JOIN Seat se
        ON se.HallId = sc.HallId
       AND se.IsActive = 1
```

```
    LEFT JOIN Ticket t
      ON t.ScreeningId = sc.ScreeningId
     AND t.SeatId = se.SeatId
     AND (
         t.Status = N'Sold'
         OR (t.Status = N'Reserved' AND t.ExpiresAt > SYSUTCDATETIME())
     )
   WHERE sc.ScreeningId = @ScreeningId
     AND t.TicketId IS NULL
);
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-04T14:38:50.7324146+03:00

Пример вызова функции

```
SELECT SeatId, RowNumber, SeatNumber
FROM GetAvailableSeats(1)
ORDER BY RowNumber, SeatNumber;
```

Результат

SeatId	RowNumber	SeatNumber
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7

Создадим табличную функцию GetCustomerTicketHistory, которая возвращает историю операций с билетами выбранного покупателя (покупки и отмены) с указанием времени операции, фильма, зала, ряда, места и стоимости билета.

Запрос

```
USE CinemaDB;
GO

CREATE FUNCTION GetCustomerTicketHistory
(
    @CustomerId INT
)
RETURNS @History TABLE
(
    TicketId INT,
    Operation NVARCHAR(20),
    OperationTime DATETIME2(0),
    MovieTitle NVARCHAR(255),
    HallId INT,
    RowNumber INT,
    SeatNumber INT,
    Price DECIMAL(10, 2)
)
AS
BEGIN
    INSERT INTO @History
        (TicketId, Operation, OperationTime, MovieTitle, HallId, RowNumber,
SeatNumber, Price)
    SELECT
    t.TicketId, N'Sold', t.SoldAt, m.Title, h.HallId, se.RowNumber, se.SeatNumber,
t.Price
    FROM Ticket AS t
    JOIN Screening AS sc ON sc.ScreeningId = t.ScreeningId
    JOIN Movie AS m  ON m.MovieId = sc.MovieId
    JOIN Hall AS h ON h.HallId = sc.HallId
```

```
    JOIN Seat AS se ON se.SeatId = t.SeatId
    WHERE t.CustomerId = @CustomerId
    AND t.Status = N'Sold'
    AND t.SoldAt IS NOT NULL;

    INSERT INTO @History
        (TicketId, Operation, OperationTime, MovieTitle, HallId, RowNumber,
SeatNumber, Price)
    SELECT
        t.TicketId, N'Cancelled', t.CancelledAt, m.Title, h.HallId, se.RowNumber,
se.SeatNumber, t.Price
    FROM Ticket AS t
    JOIN Screening AS sc ON sc.ScreeningId = t.ScreeningId
    JOIN Movie AS m ON m.MovieId = sc.MovieId
    JOIN Hall AS h ON h.HallId = sc.HallId
    JOIN Seat AS se ON se.SeatId = t.SeatId
    WHERE t.CustomerId = @CustomerId
    AND t.Status = N'Cancelled'
    AND t.CancelledAt IS NOT NULL;

    RETURN;
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-04T14:54:24.2698674+03:00

Пример вызова функции

```
USE CinemaDB;
GO
```

```
SELECT
   TicketId, Operation, OperationTime, MovieTitle,
   HallId, RowNumber, SeatNumber, Price
FROM GetCustomerTicketHistory(1)
ORDER BY OperationTime;
```

Результат

Результаты | Сообщения

	TicketId	Operation	OperationTime	MovieTitle	HallId	RowNumber	SeatNumber	Price
1	6	Sold	2025-11-30 11:57:00	Матрица	4	1	1	500.00

Рисунок 3

Представления (VIEW)

Создадим представление ScreeningSeatMap, которое формирует карту мест по сеансам с указанием статуса каждого места (свободно, забронировано, продано, отключено).

Запрос

```
USE CinemaDB;
GO

CREATE VIEW ScreeningSeatMap
AS
SELECT
  sc.ScreeningId,
  sc.StartTime,
  m.Title AS MovieTitle,
  h.HallId,
  h.Type AS HallType,
  se.SeatId,
  se.RowNumber,
  se.SeatNumber,
  CASE
```

```
    WHEN se.IsActive = 0 THEN N'Disabled'
    WHEN t.Status = N'Sold' THEN N'Sold'
    WHEN t.Status = N'Reserved'
        AND t.ExpiresAt IS NOT NULL
        AND t.ExpiresAt > SYSUTCDATETIME() THEN N'Reserved'
    ELSE N'Free'
  END AS SeatStatus
FROM Screening sc
JOIN Movie m  ON m.MovieId = sc.MovieId
JOIN Hall h  ON h.HallId  = sc.HallId
JOIN Seat se ON se.HallId = sc.HallId
LEFT JOIN Ticket t
  ON t.ScreeningId = sc.ScreeningId
 AND t.SeatId = se.SeatId;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T19:51:16.7542926+03:00

Использование представления

```
SELECT ScreeningId, RowNumber, SeatNumber, SeatStatus
FROM ScreeningSeatMap
WHERE ScreeningId = 1
ORDER BY RowNumber, SeatNumber;
```

Результат

ScreeningId	RowNumber	SeatNumber	SeatStatus
1	1	1	Sold
1	1	2	Sold
1	1	3	Free
1	1	4	Free
1	1	5	Free

… … … …

Создадим представление UpcomingScreenings, которое отображает список ближайших сеансов с информацией о фильме, зале, базовой цене, количестве свободных мест и текущей выручке.

Запрос

```
USE CinemaDB;
GO

CREATE VIEW UpcomingScreenings
AS
SELECT
  sc.ScreeningId,
  sc.StartTime,
  m.Title AS MovieTitle,
  m.AgeRating,
  m.DurationMin,
  h.HallId,
  h.Type AS HallType,
  sc.BasePrice,
  GetAvailableSeatsCount(sc.ScreeningId) AS AvailableSeats,
  GetScreeningRevenue(sc.ScreeningId) AS Revenue
FROM Screening sc
JOIN Movie m ON m.MovieId = sc.MovieId
JOIN Hall  h ON h.HallId  = sc.HallId
WHERE sc.StartTime >= SYSUTCDATETIME();
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T19:55:39.2383106+03:00

Использование представления

```
USE CinemaDB;
GO

SELECT ScreeningId, StartTime, MovieTitle, AgeRating, DurationMin, HallId, HallType, BasePrice, AvailableSeats, Revenue
FROM UpcomingScreenings
ORDER BY StartTime;
```

Результат

Результаты | Сообщения

	ScreeningId	StartTime	MovieTitle	AgeRating	DurationMin	HallId	HallType	BasePrice	AvailableSeats	Revenue
1	4	2025-12-01 19:00:00	Матрица	16+	136	4	IMAX	500.00	0	500.00

Рисунок 4

Создадим представление CustomerTickets, которое объединяет данные о билетах с информацией о покупателях, сеансах, фильмах и местах для формирования сводной отчётности.

Запрос

```
USE CinemaDB;
GO

CREATE VIEW CustomerTickets
AS
SELECT
  t.TicketId,
  c.CustomerId,
  c.FullName AS CustomerName,
  c.Email,
  t.Status,
```

```
  t.Price,
  t.CreatedAt,
  t.SoldAt,
  t.CancelledAt,
  sc.ScreeningId,
  sc.StartTime,
  m.Title AS MovieTitle,
  h.HallId,
  h.Type AS HallType,
  se.RowNumber,
  se.SeatNumber
FROM Ticket AS t
JOIN Customer AS c ON c.CustomerId = t.CustomerId
JOIN Screening AS sc ON sc.ScreeningId = t.ScreeningId
JOIN Movie AS m ON m.MovieId = sc.MovieId
JOIN Hall AS h ON h.HallId = sc.HallId
JOIN Seat AS se ON se.SeatId = t.SeatId;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-11-30T20:09:39.1783168+03:00

Использование представления

```
USE CinemaDB;
GO

SELECT TicketId, CustomerId, CustomerName, Email, Status, Price, CreatedAt,
SoldAt, CancelledAt, ScreeningId, StartTime, MovieTitle, HallId, HallType,
RowNumber, SeatNumber
FROM CustomerTickets
WHERE MovieTitle = N'Интерстеллар'
```

```
ORDER BY StartTime;
```

Результат

Результаты | Сообщения

	TicketId	CustomerId	CustomerName	Email	Status	Price	CreatedAt	SoldAt	CancelledAt	ScreeningId	StartTime	MovieTitle	HallId	HallType	RowNumber	SeatNumber
1	1	1	Иванов Иван Иванович	ivanov@example.com	Sold	350.00	2025-11-20 09:53:53	NULL	NULL	1	2025-11-21 18:00:00	Интерстеллар	1	STANDARD	1	1
2	2	2	Петров Пётр Петрович	petrov@example.com	Sold	350.00	2025-11-20 09:53:53	NULL	NULL	1	2025-11-21 18:00:00	Интерстеллар	1	STANDARD	1	2
3	3	2	Петров Пётр Петрович	petrov@example.com	Reserved	350.00	2025-11-20 09:53:53	NULL	NULL	1	2025-11-21 18:00:00	Интерстеллар	1	STANDARD	1	3

Рисунок 5

13.Триггеры

Создадим триггер trigger_Ticket_Audit, который будет срабатывать после вставки, изменения или удаления строк в таблице Ticket и записывать информацию об операции в таблицу TicketAudit

Запрос

```
USE CinemaDB;
GO

CREATE TRIGGER trigger_Ticket_Audit
ON Ticket
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
   INSERT INTO TicketAudit (TicketId, Operation, OldStatus, NewStatus,
UserName)
   SELECT
      COALESCE(d.TicketId, i.TicketId) AS TicketId,
      CASE
         WHEN d.TicketId IS NULL THEN N'INSERT'
         WHEN i.TicketId IS NULL THEN N'DELETE'
         ELSE N'UPDATE'
      END AS Operation,
      d.Status AS OldStatus,
      i.Status AS NewStatus,
      ORIGINAL_LOGIN() AS UserName
   FROM inserted i
   FULL OUTER JOIN deleted d ON d.TicketId = i.TicketId;
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-03T19:53:59.6732281+03:00

Проверка срабатывания триггера

```
USE CinemaDB;
GO

UPDATE Ticket
SET Status = N'Cancelled'
WHERE TicketId = 1;

SELECT TOP 5 AuditId, TicketId, Operation, OldStatus, NewStatus, ChangeDate,
UserName
FROM TicketAudit
ORDER BY ChangeDate DESC;
```

Результат

Результаты | Сообщения

	AuditId	TicketId	Operation	OldStatus	NewStatus	ChangeDate	UserName
1	1	1	UPDATE	Sold	Cancelled	2025-12-03 16:58:31	MicrosoftAccount\bricking5654help@yandex.ru

Рисунок 6

Создадим триггер trigger_Ticket_NoDobuleBooking

Запрос

```
USE CinemaDB;
GO

CREATE TRIGGER trigger_Ticket_NoDoubleBooking
ON Ticket
AFTER INSERT, UPDATE
AS
BEGIN
```

```
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Ticket t
          ON t.ScreeningId = i.ScreeningId
         AND t.SeatId = i.SeatId
         AND t.TicketId <> i.TicketId
        WHERE t.Status IN (N'Reserved', N'Sold')
          AND i.Status IN (N'Reserved', N'Sold')
    )
    BEGIN
        RAISERROR(
            N'Невозможно сохранить запись: место уже занято (продано или находится в активной броне) для данного сеанса.',
            16, 1
        );
        ROLLBACK TRANSACTION;
        RETURN;
    END;
END;
GO
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-03T21:02:48.3969057+03:00

Проверка срабатывания триггера

```
USE CinemaDB;
GO

INSERT INTO Ticket (ScreeningId, SeatId, CustomerId, Status, ExpiresAt, Price)
```

```
VALUES (1, 2, 1, N'Reserved', DATEADD(MINUTE, 10, SYSUTCDATETIME()), 350.00);
```

Результат

(Затронута 1 строка)

сообщение: 50000, уровень: 16, состояние: 1, процедура: trigger_Ticket_NoDoubleBooking, строка: 18 [строка начала пакета: 2]

Невозможно сохранить запись: место уже занято (продано или находится в активной броне) для данного сеанса.

Сообщение 3609, уровень 16, состояние 1, строка 4

Транзакция завершилась в триггере. Выполнение пакета прервано.

Время выполнения: 2025-12-03T21:06:28.5402012+03:00

Создадим триггер trigger_TicketSeatHallMatch, который проверяет соответствие зала между местом и сеансом

Запрос

```
USE CinemaDB;
GO

CREATE TRIGGER trigger_Ticket_SeatHallMatch
ON Ticket
AFTER INSERT, UPDATE
AS
BEGIN
  IF EXISTS (
    SELECT 1
    FROM inserted i
    JOIN Screening s ON s.ScreeningId = i.ScreeningId
    JOIN Seat      st ON st.SeatId      = i.SeatId
    WHERE s.HallId <> st.HallId
  )
```

```
    BEGIN
        RAISERROR (N'Зал сеанса и места должны совпадать', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END;
```

Результат

Выполнение команд успешно завершено.

Время выполнения: 2025-12-03T23:36:33.3864016+03:00

Проверка срабатывания триггера

```
USE CinemaDB;
GO

DECLARE @ScreeningId INT;
DECLARE @SeatId      INT;
DECLARE @CustomerId  INT;

SET @ScreeningId = 1;   -- ScreeningId = 1, HallId = 1
SET @SeatId = 51;  -- SeatId = 51, HallId = 2
SET @CustomerId = 1;

INSERT INTO Ticket (ScreeningId, SeatId, CustomerId, Status, ExpiresAt, Price)
VALUES (
    @ScreeningId,
    @SeatId,
    @CustomerId,
    N'Reserved',
    DATEADD(MINUTE, 10, SYSUTCDATETIME()),
    350.00
);
```

Результат

сообщение: 50000, уровень: 16, состояние: 1, процедура: trigger_Ticket_SeatHallMatch, строка: 15 [строка начала пакета: 2]

Зал сеанса и места должны совпадать

Сообщение 3609, уровень 16, состояние 1, строка 12

Транзакция завершилась в триггере. Выполнение пакета прервано.

Время выполнения: 2025-12-04T00:07:19.0145556+03:00

14. Стратегия резервного копирования

Цели резервного копирования

- сохранения информации о сеансах, продажах билетов и клиентах при любых сбоях оборудования или ПО
- возможность оперативно восстановить работу системы онлайн-продаж и касс
- минимизации потери данных

Наиболее важными данными для CinemaDB считаем

- Сведения о клиентах
- Каталог фильмов и сеансов
- Информация о билетах и фактах продажи

Поскольку потеря этих данных приведет к невозможности продажи, бронированию и возврату билетов

Типы резервных копий и расписание

1. Полное резервное копирование
   - Периодичность: один раз в неделю
   - Время: ночь с субботы на воскресенье в 2:00
   - Срок хранения: 30 дней
2. Дифференциальное резервное копирование
   - Периодичность: каждый день, кроме дня, когда делается полный бэкап
   - Время: 21:00
   - Срок хранения: 7 дней
3. Резервное копирование журнала транзакций
   - Периодичность: каждые 2 часа в рабочее время кинотеатра
   - Срок хранения

Процедура восстановления

При полном сбое:

- Восстановить последнее полное резервное копирование
- Применить последднее дифференциальное резервное копирование
- Применить все из резервного копирования журнала транзакций

При частичной потере данных:

- Восстановить все из резервного копирования журнала транзакций

15.Создание резервного копирования

Полное резервное копирование

Запрос

```
BACKUP DATABASE CinemaDB
TO DISK = 'C:\GitHub\course-3\db\idz\BACKUPS\CinemaDB_Full.bak'
WITH
   NAME = 'CinemaDB_Full_Backup',
   DESCRIPTION = N'Полная резервная копия базы данных кинотеатра CinemaDB',
   FORMAT,
   STATS = 10,
   CHECKSUM;
GO
```

Результат

10 проц. обработано.

20 проц. обработано.

…

90 проц. обработано.

100 проц. обработано.

Обработано 696 страниц для базы данных "CinemaDB", файл "CinemaDB" для файла 1.

Обработано 1 страниц для базы данных "CinemaDB", файл "CinemaDB_log" для файла 1.

BACKUP DATABASE успешно обработал 697 страниц за 0.074 секунд (73.499 МБ/сек).

Время выполнения: 2025-11-30T20:44:29.2763838+03:00

Дифференциальное резервное копирование

Запрос

```
BACKUP DATABASE CinemaDB
```

```
TO DISK = 'C:\GitHub\course-3\db\idz\BACKUPS\CinemaDB_Diff.bak'
WITH
   DIFFERENTIAL,
   NAME = 'CinemaDB_Diff_Backup',
   DESCRIPTION = N'Дифференциальная резервная копия базы данных
CinemaDB',
   INIT,
   STATS = 10,
   CHECKSUM;
GO
```

Результат

17 проц. обработано.

26 проц. обработано.

…

98 проц. обработано.

100 проц. обработано.

Обработано 104 страниц для базы данных "CinemaDB", файл "CinemaDB" для файла 1.

Обработано 1 страниц для базы данных "CinemaDB", файл "CinemaDB_log" для файла 1.

BACKUP DATABASE WITH DIFFERENTIAL успешно обработал 105 страниц за 0.024 секунд (33.915 МБ/сек).

Время выполнения: 2025-11-30T20:45:40.0595659+03:00

Резервное копирование журнала транзацкий

Запрос

```
USE master;
GO

BACKUP LOG CinemaDB
```

```
TO DISK = 'C:\GitHub\course-
3\db\idz\BACKUPS\CinemaDB_Log_20251130_2045.trn'
WITH
   INIT,
   NAME = N'CinemaDB_log_30_11_2025_20_45',
   STATS = 10;
GO
```

Результат

100 проц. обработано.

Обработано 1 страниц для базы данных "CinemaDB", файл "CinemaDB_log" для файла 1.

BACKUP LOG успешно обработал 1 страниц за 0.003 секунд (2.115 МБ/сек).

Время выполнения: 2025-12-03T21:29:32.4513912+03:00

16.Восстановление базы данных

Восстанавливаем базу данных с помощью bak файлов, предварительно удалив БД.

Восстановление последнего полного резервного копирования

Запрос

```
RESTORE DATABASE CinemaDB
FROM DISK = 'C:\GitHub\course-3\db\idz\BACKUPS\CinemaDB_Full.bak'
WITH
   NORECOVERY,
   REPLACE,
   STATS = 10;
```

Результат

10 проц. обработано.

20 проц. обработано.

…

90 проц. обработано.

100 проц. обработано.

Обработано 696 страниц для базы данных "CinemaDB", файл "CinemaDB" для файла 1.

Обработано 1 страниц для базы данных "CinemaDB", файл "CinemaDB_log" для файла 1.

RESTORE DATABASE успешно обработал 697 страниц за 0.024 секунд (226.623 МБ/сек).

Время выполнения: 2025-12-03T21:15:05.7516569+03:00

Восстановление последнего дифференциального резервного копирования

Запрос

```
RESTORE DATABASE CinemaDB
FROM DISK = 'C:\GitHub\course-3\db\idz\BACKUPS\CinemaDB_Diff.bak'
WITH
   NORECOVERY,
   STATS = 10;
```

Результат

17 проц. обработано.

26 проц. обработано.

…

98 проц. обработано.

100 проц. обработано.

Обработано 104 страниц для базы данных "CinemaDB", файл "CinemaDB" для файла 1.

Обработано 1 страниц для базы данных "CinemaDB", файл "CinemaDB_log" для файла 1.

RESTORE DATABASE успешно обработал 105 страниц за 0.011 секунд (73.996 МБ/сек).

Время выполнения: 2025-12-03T21:16:01.8281915+03:00

Восстановление журнала транзакций

Запрос

```
RESTORE LOG CinemaDB
FROM DISK = 'C:\GitHub\course-3\db\idz\BACKUPS\CinemaDB_Log_20251130_2045.trn'
WITH
   RECOVERY,
   STATS = 10;
```

Результат

100 проц. обработано.

Обработано 0 страниц для базы данных "CinemaDB", файл "CinemaDB" для файла 1.

Обработано 6 страниц для базы данных "CinemaDB", файл "CinemaDB_log" для файла 1.

RESTORE LOG успешно обработал 6 страниц за 0.004 секунд (11.596 МБ/сек).

Время выполнения: 2025-12-03T21:58:55.5452441+03:00

Проверим корректность восстановленных данных

Запрос

```
USE CinemaDB;
GO
SELECT CustomerId, FullName, Email, Phone, RegisteredAt
```

FROM Customer

Результат

Результаты | Сообщения

	CustomerId	FullName	Email	Phone	RegisteredAt
1	1	Иванов Иван Иванович	ivanov@example.com	+79001112233	2025-11-20 09:22:48
2	2	Петров Пётр Петрович	petrov@example.com	+79002223344	2025-11-20 09:22:48
3	3	Сидорова Анна Сергеевна	sidorova@example.com	+79003334455	2025-11-20 09:22:48
4	4	Кузнецов Алексей Сергеевич	akuznetsov@example.com	+79004445566	2025-11-30 11:13:21

Рисунок 7

17.Заключение

В рамках курсовой работы была спроектирована и реализована база данных CinemaDB, предназначенная для поддержки работы кинотеатра: хранения информации о фильмах, залах, местах, сеансах и продаже билетов зарегистрированным клиентам.

Структура базы данных включает основные таблицы: Customer, Movie, Hall, Seat, Screening и Ticket, TicketAudit. Между таблицами заданы связи через внешние ключи, что гарантирует согласованность данных (например, билет не может ссылаться на несуществующий сеанс, место или покупателя). Ограничения CHECK используются для проверки бизнес-правил: допустимые возрастные рейтинги фильмов (0+, 6+, 12+, 16+, 18+), ненулевые и неотрицательные цены, положительная длительность фильма, корректные номера рядов и мест. Для некоторых полей дополнительно введены уникальные ограничения, исключающие повторения значений в поле.

Для работы с данными и бизнес-логикой продажи билетов созданы хранимые процедуры: AddCustomer, AddMovie, AddHall, AddSeat, AddScreening, а также процедуры ReserveTicket, SellTicket и CancelReservation, реализующие сценарии бронирования, продажи и отмены брони. Внутри процедур используются транзакции и обработка ошибок, что позволяет корректно реагировать на нарушения ограничений (например, попытку продать уже занятое место) и не оставлять базу в некорректном состоянии. Пользовательские функции GetAvailableSeatsCount, GetScreeningRevenue и обеспечивают простое получение аналитической информации: количество свободных мест на сеанс, выручка по конкретному показу и число купленных билетов у выбранного клиента. Дополнительно реализованы триггеры trigger_Ticket_Audit, trigger_Ticket_NoDoubleBooking и trigger_Ticket_SeatHallMatch, которые автоматически контролируют корректность операций с билетами (предотвращают двойное бронирование и несоответствие зала места и сеанса) и журналируют все изменения статусов билетов в таблице TicketAudit.

Для удобного просмотра данных и формирования отчётов созданы представления ScreeningSeatMap (карта занятости мест по сеансам), UpcomingScreenings (список ближайших показов с указанием фильмов, залов, базовой цены, выручки и числа свободных мест) и CustomerTickets (сводная информация по билетам с привязкой к клиентам, фильмам, сеансам и местам). Эти представления демонстрируют корректную работу соединений между таблицами и упрощают построение запросов для анализа.

Отдельно описана и реализована стратегия резервного копирования базы данных CinemaDB. Предусмотрены регулярные полные резервные копии, дифференциальные бэкапы и сохранение журнала транзакций, что позволяет восстановить базу как после серьёзного сбоя, так и до нужного момента времени при логических ошибках. Проверка выполнения команд BACKUP DATABASE и BACKUP LOG показала корректность создания резервных копий. В итоге получена работоспособная база данных, которая решает поставленные задачи: ведёт учёт клиентов, фильмов, залов, сеансов и продаж билетов, поддерживает операции бронирования и оплаты, а также обеспечивает надёжное хранение и восстановление информации.

18. Список использованных источников

1. Горячев А. В., Новакова Н. Е. Распределенные базы данных. Мет. указания к лаб. работам., СПб. Изд-во СПбГЭТУ «ЛЭТИ», 2008
2. Горячев А. В., Новакова Н.Е. Особенности разработки и администрирования приложений баз данных: учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2016. 68 с.
3. Веб-сервис для создания диаграмм https://www.drawio.com/
4. Онлайн-курс «Основы баз данных SQL» [Электронный ресурс]: https://start.practicum.yandex/sql-database-basics/
5. Документация Transact-SQL: оператор сравнения «не равно» (<>). Microsoft Learn [Электронный ресурс]: https://learn.microsoft.com/ru-ru/sql/t-sql/language-elements/not-equal-to-transact-sql-traditional?view=sql-server-ver17
6. Документация Transact-SQL: Резервное копирование. Microsoft Learn [Электронный ресурс]: https://learn.microsoft.com/ru-ru/sql/t-sql/statements/backup-transact-sql?view=sql-server-ver17