-- Создание базы данных

CREATE DATABASE Кинотеатр;

GO

-- Использование базы данных

USE Кинотеатр;

GO

-- Создание таблицы "Фильмы"

CREATE TABLE Фильмы (

id\_фильма INT PRIMARY KEY IDENTITY(1, 1),

Название VARCHAR(100) NOT NULL,

Режиссер VARCHAR(100) NOT NULL,

Год\_выпуска INT NOT NULL

);

GO

-- Создание таблицы "Залы"

CREATE TABLE Залы (

id\_зала INT PRIMARY KEY IDENTITY(1, 1),

Название VARCHAR(100) NOT NULL,

Количество\_мест INT NOT NULL

);

GO

-- Создание таблицы "Сеансы"

CREATE TABLE Сеансы (

id\_сеанса INT PRIMARY KEY IDENTITY(1, 1),

id\_фильма INT,

id\_зала INT,

Дата date,

Время time,

FOREIGN KEY (id\_фильма) REFERENCES Фильмы (id\_фильма),

FOREIGN KEY (id\_зала) REFERENCES Залы (id\_зала)

);

GO

-- Создание таблицы "Билеты"

CREATE TABLE Билеты (

id\_билета INT PRIMARY KEY IDENTITY(1, 1),

id\_сеанса INT,

Ряд INT NOT NULL,

Место INT NOT NULL,

FOREIGN KEY (id\_сеанса) REFERENCES Сеансы (id\_сеанса)

);

GO

-- Заполнение таблицы "Фильмы"

INSERT INTO Фильмы (Название, Режиссер, Год\_выпуска)

VALUES

('Фильм 1', 'Режиссер 1', 2021),

('Фильм 2', 'Режиссер 2', 2022),

('Фильм 3', 'Режиссер 3', 2023);

-- Заполнение таблицы "Залы"

INSERT INTO Залы (Название, Количество\_мест)

VALUES

('Зал 1', 100),

('Зал 2', 150),

('Зал 3', 200);

-- Заполнение таблицы "Сеансы"

INSERT INTO Сеансы (id\_фильма, id\_зала, Дата, Время)

VALUES

(1, 1, '2023-06-07', '10:00:00'),

(2, 2, '2023-06-08', '14:00:00'),

(3, 3, '2023-06-06', '18:00:00');

-- Заполнение таблицы "Билеты"

INSERT INTO Билеты (id\_сеанса, Ряд, Место)

VALUES

(1, 1, 1),

(1, 1, 2),

(1, 2, 3),

(2, 1, 1),

(2, 2, 2),

(2, 3, 3),

(3, 1, 1),

(3, 2, 2),

(3, 3, 3);

CREATE VIEW Сеансы\_Фильмы\_Залы AS

SELECT Сеансы.id\_сеанса, Фильмы.Название AS Фильм, Залы.Название AS Зал, Сеансы.Дата, Сеансы.Время

FROM Сеансы

JOIN Фильмы ON Сеансы.id\_фильма = Фильмы.id\_фильма

JOIN Залы ON Сеансы.id\_зала = Залы.id\_зала;

SELECT \* FROM Сеансы\_Фильмы\_Залы;

CREATE VIEW Фильмы\_Режиссеры AS

SELECT Фильмы.Название AS Фильм, Фильмы.Режиссер

FROM Фильмы;

SELECT \* FROM Фильмы\_Режиссеры;

CREATE VIEW Свободные\_Места\_Залы AS

SELECT Залы.id\_зала, Залы.Название, Залы.Количество\_мест - COUNT(Билеты.id\_билета) AS Свободные\_места

FROM Залы

LEFT JOIN Сеансы ON Залы.id\_зала = Сеансы.id\_зала

LEFT JOIN Билеты ON Сеансы.id\_сеанса = Билеты.id\_сеанса

GROUP BY Залы.id\_зала, Залы.Название, Залы.Количество\_мест;

SELECT \* FROM Свободные\_Места\_Залы;

CREATE PROCEDURE ДобавитьФильм

@Название VARCHAR(100),

@Режиссер VARCHAR(100),

@Год\_выпуска INT

AS

BEGIN

INSERT INTO Фильмы (Название, Режиссер, Год\_выпуска)

VALUES (@Название, @Режиссер, @Год\_выпуска);

END;

CREATE PROCEDURE УдалитьФильм

@id\_фильма INT

AS

BEGIN

DELETE FROM Фильмы

WHERE id\_фильма = @id\_фильма;

END;

CREATE PROCEDURE ОбновитьФильм

@id\_фильма INT,

@Название VARCHAR(100),

@Режиссер VARCHAR(100),

@Год\_выпуска INT

AS

BEGIN

UPDATE Фильмы

SET Название = @Название,

Режиссер = @Режиссер,

Год\_выпуска = @Год\_выпуска

WHERE id\_фильма = @id\_фильма;

END;

--Триггер для удаления связанных билетов при удалении сеанса:

CREATE TRIGGER Удаление\_связанных\_билетов

ON Сеансы

AFTER DELETE

AS

BEGIN

DELETE FROM Билеты

WHERE id\_сеанса IN (SELECT id\_сеанса FROM deleted);

END;

--". Он объединяет таблицы "Билеты" и "Сеансы", чтобы проверить, что сеанс, на который были проданы билеты, активен

--(то есть его дата и время больше текущего времени).

CREATE TRIGGER CheckActiveSessions

ON Билеты

AFTER INSERT

AS

BEGIN

IF EXISTS (SELECT \* FROM inserted i JOIN Сеансы s ON i.id\_сеанса = s.id\_сеанса WHERE s.Дата < (SELECT CAST(GETDATE() AS DATE) AS CurrentDateOnly) OR s.Время < (SELECT CONVERT(TIME, GETDATE()) AS CurrentTime))

BEGIN

ROLLBACK TRANSACTION

PRINT 'Нельзя продать билеты на неактивный сеанс!'

END

END

--будет выводить информацию о вставленном сеансе.

CREATE TRIGGER Вывод\_информации\_о\_сеансе

ON Сеансы

AFTER INSERT

AS

BEGIN

DECLARE @id\_сеанса INT, @id\_фильма INT, @id\_зала INT, @Дата DATE, @Время TIME;

SELECT @id\_сеанса = id\_сеанса, @id\_фильма = id\_фильма, @id\_зала = id\_зала, @Дата = Дата, @Время = Время

FROM inserted;

PRINT 'Добавлен новый сеанс:';

PRINT 'ID сеанса: ' + CAST(@id\_сеанса AS VARCHAR(10));

PRINT 'ID фильма: ' + CAST(@id\_фильма AS VARCHAR(10));

PRINT 'ID зала: ' + CAST(@id\_зала AS VARCHAR(10));

PRINT 'Дата: ' + CONVERT(VARCHAR(10), @Дата, 101);

PRINT 'Время: ' + CONVERT(VARCHAR(8), @Время, 108);

END;

CREATE FUNCTION ПолучитьСписокФильмов()

RETURNS TABLE

AS

RETURN

(

SELECT id\_фильма, Название, Режиссер, Год\_выпуска

FROM Фильмы

);

CREATE FUNCTION ПолучитьСписокСеансовПоФильму(@id\_фильма INT)

RETURNS TABLE

AS

RETURN

(

SELECT id\_сеанса, Дата, Время, Название, Количество\_мест

FROM Сеансы

INNER JOIN Залы ON Сеансы.id\_зала = Залы.id\_зала

WHERE Сеансы.id\_фильма = @id\_фильма

);

CREATE FUNCTION ПолучитьСписокЗанятыхМестВСеансе(@id\_сеанса INT)

RETURNS TABLE

AS

RETURN

(

SELECT Ряд, Место

FROM Билеты

WHERE id\_сеанса = @id\_сеанса

);

1. **Дайте определение структуры данных СУБД, общий подход к организации представлений, таблиц, индексов и кластеров.**

Структура данных в СУБД (системе управления базами данных) определяет организацию и хранение данных в базе данных. Она определяет формат данных, способ их хранения, связи между данными и доступ к ним. Структура данных включает в себя следующие элементы:

1. Таблицы: Таблица представляет собой структуру данных, состоящую из строк и столбцов.

2. Поля: Поля представляют собой отдельные элементы данных в таблице. Они определяют тип данных (например, целые числа, строки, даты), ограничения, форматирование и другие атрибуты данных.

3. Ключи: Ключи используются для уникальной идентификации записей в таблице. Основным ключом (Primary Key) является уникальный идентификатор каждой записи, который обеспечивает идентификацию и целостность данных. Вторичные ключи (Secondary Key) могут использоваться для быстрого доступа к данным по определенным критериям.

4. Связи: Связи определяют отношения и связи между таблицами. Они позволяют объединять данные из разных таблиц, основываясь на общих полях или ключах.

5. Индексы: Индексы создаются для оптимизации производительности поиска и сортировки данных. Они создаются на одном или нескольких полях таблицы и позволяют быстрый доступ к данным по этим полям.

Общий подход к организации представлений, таблиц, индексов и кластеров в базе данных включает следующие принципы:

1. Правильное проектирование схемы базы данных: перед созданием таблиц и других структур данных необходимо провести анализ требований и разработать соответствующую схему базы данных. Это включает определение сущностей, их атрибутов, связей и правил целостности данных.

2. Нормализация таблиц: Нормализация является процессом разделения таблиц на более мелкие и связанные сущности, чтобы устранить избыточность данных и обеспечить целостность. Нормализация помогает предотвратить дублирование данных и обеспечить эффективность операций обновления и запросов.

3. Создание индексов: Индексы используются для ускорения выполнения операций поиска и сортировки. Они создаются на ключевых полях таблицы, которые часто используются в запросах. Оптимальный подход к созданию индексов включает анализ запросов и выбор ключевых полей, на которых индексы будут наиболее эффективны.

4. Кластеризация данных: Кластеризация определяет физическую организацию данных на диске. Целью кластеризации является логическое и физическое размещение связанных данных рядом друг с другом. Кластеризация может быть основана на ключевых полях или других критериях. Оптимальное размещение данных на диске может существенно повысить производительность операций чтения и записи.

5. Эффективное использование представлений: Представления представляют собой виртуальные таблицы, которые основаны на данных из одной или нескольких таблиц. Они используются для упрощения доступа к данным и скрытия сложности физической структуры базы данных.