Лабораторная работа 14

До подступления к основной части, хочу отметить Как изменилась структура базы после сдачи ЛР12

- рекурсивная связь – когда таблица ссылается сама на себя. как таковых рекурсивных связей у меня нет. рекурсивных циклов тоже не наблюдается.

- М:М продавец диск – да, это логически не верно (не может быть так, чтобы один диск был одновременно в аренде у двух фирм). поэтому – нужно было что-либо менять.

с учетом цели для таблицы, которую я ей ставил в начале: когда взят в аренду – добавляем в таблицу, когда сдан обратно – удаляем, мы не храним в ней никакой истории, мы не сдаем в аренду. был вариант хранить все в Carrier (null – значит не в аренду), но я посчитал это достаточно избыточным и решил обойтись все той же таблицей И сделать в качестве Primary Key сам Carrier\_id.

но если бы я так сделал, то потерял бы в производительности в ряде случаев (к примеру: чтобы запросить все Свои фильмы, мне необходимо было бы и запросить все id из таблицы аренды и оно бы так долго проверяло)

и мне не нужно экономить в памяти – у меня не самая большая домашняя видеотека, я не потеряю много лишней памяти на пустых полях даты и арендадателя, чтобы так за это переживать.

поэтому

- в сам диск вставить аренду – я принял это решение

- может убрать userfavorite – я решил их оставить, так как они носят ту логическую функцию, которую не несет статистика по WatchHistory

в итоге все выглядит так

CREATE TABLE [Carrier] (

[id] INT PRIMARY KEY IDENTITY(1, 1),

[size] INT CHECK ([size] >= 0), -- 0 для облачных сервисов

[name] VARCHAR(255) NOT NULL,

[type\_id] INT,

[start\_rent] DATETIME,

[end\_rent] DATETIME,

[rentaler\_concrete\_id] INT

)

GO

ALTER TABLE [Carrier] ADD FOREIGN KEY ([type\_id]) REFERENCES [Type] ([id])

GO

ALTER TABLE [Carrier] ADD FOREIGN KEY ([rentaler\_concrete\_id]) REFERENCES [Rentaler\_Concrete] ([id])

GO

INSERT INTO Carrier (id, size, name, type\_id, start\_rent, end\_rent, rentaler\_concrete\_id) VALUES

-- просрочил

(1, 700, 'Carrier A', 1,DATEADD(DAY, -10, GETDATE()), DATEADD(DAY, -9, GETDATE()), 1),

(2, 2500, 'Carrier B', 2, DATEADD(DAY, -8, GETDATE()), DATEADD(DAY, -7, GETDATE()), 2),

(3, 1200, 'Carrier C', 3, DATEADD(DAY, -6, GETDATE()), DATEADD(DAY, -5, GETDATE()), 3),

(4, 0, 'Digital Stream', 4, DATEADD(DAY, -4, GETDATE()), DATEADD(DAY, -3, GETDATE()), 4),

-- сегодня закончиться

(5, 4400, 'LaserDisc 1', 5, DATEADD(DAY, -1, GETDATE()), DATEADD(HOUR, 2, GETDATE()), 5),

-- активные

(6, 800, 'Betamax Tape', 6, DATEADD(DAY, -2, GETDATE()), DATEADD(DAY, 1, GETDATE()), 6),

(7, 5000, 'HD DVD 1', 7, DATEADD(DAY, -3, GETDATE()), DATEADD(DAY, 2, GETDATE()), 7),

(8, 6000, '4K UHD Disk', 8, DATEADD(DAY, -4, GETDATE()), DATEADD(DAY, 3, GETDATE()), 8),

(9, 650, 'Audio CD', 9, DATEADD(DAY, -5, GETDATE()), DATEADD(DAY, 4, GETDATE()), 9),

(10, 90, 'Music Cassette', 10, DATEADD(DAY, -6, GETDATE()), DATEADD(DAY, 5, GETDATE()), 10),

-- и тд

(11, 0, 'Netflix Stream', 11, NULL, NULL, NULL),

(12, 16000, 'USB Stick', 12, NULL, NULL, NULL),

(13, 100000, 'External HDD 1', 13, NULL, NULL, NULL),

(14, 64, 'SD Card 64GB', 14, NULL, NULL, NULL),

(15, 2700, 'Blu-ray 3D Disc', 15, NULL, NULL, NULL),

(16, 2700, 'Blu-ray with Intersteller', 15, NULL, NULL, NULL),

(17, 3700, 'Blu-ray with La La Land', 15, NULL, NULL, NULL),

(18, 2500, 'Blu-ray withThe Shawshank Redemption', 15, NULL, NULL, NULL),

(19, 2900, 'Blu-ray with Spirited Away', 15, NULL, NULL, NULL),

(20, 3000, 'Blu-ray with The Grand Budapest Hotel', 15, NULL, NULL, NULL),

(21, 700, 'Collectors DVD', 6, GETDATE(), DATEADD(DAY, 1, GETDATE()), 11),

(22, 2500, 'Directors Blu-ray', 6, NULL, NULL, NULL),

(23, 700, 'Limited VHS', 2, GETDATE(), DATEADD(DAY, 1, GETDATE()), 9),

(24, 16000, 'HD Stream', 1, NULL, NULL, NULL),

(25, 0, 'LaserClassic', 11, GETDATE(), DATEADD(DAY, 5, GETDATE()), 8),

(26, 700, 'BetaArchive', 15, NULL, NULL, NULL),

(27, 0, 'HD Master Copy', 6, GETDATE(), DATEADD(DAY, 4, GETDATE()), 11),

(28, 700, '4K Collector Edition', 11, NULL, NULL, NULL),

(29, 700, 'Studio CD', 14, GETDATE(), DATEADD(DAY, 3, GETDATE()), 7),

(30, 0, 'Classic Cassette', 9, NULL, NULL, NULL),

(31, 700, 'Netflix Digital', 11, GETDATE(), DATEADD(DAY, 2, GETDATE()), 13),

(32, 700, 'USB Archive', 4, NULL, NULL, NULL),

(33, 5000, 'HDD Vault', 7, GETDATE(), DATEADD(DAY, 2, GETDATE()), 6),

(34, 5000, 'SD Memory', 10, NULL, NULL, NULL),

(35, 5000, 'Blu-ray 3D Master', 13, GETDATE(), DATEADD(DAY, 1, GETDATE()), 11),

(36, 2500, 'Bonus Disk', 1, NULL, NULL, NULL),

(37, 2500, 'Behind the Scenes', 9, GETDATE(), DATEADD(DAY, 5, GETDATE()), 11),

(38, 5000, 'Remastered Copy', 10, NULL, NULL, NULL),

(39, 0, 'Studio Screener', 5, GETDATE(), DATEADD(DAY, 3, GETDATE()), 7),

(40, 0, 'Original Print', 13, NULL, NULL, NULL);

-- 6

-- список носителей и статус их просроченности

SELECT

c.name AS carrier\_name,

c.end\_rent, -- Берем дату окончания аренды прямо из таблицы Carrier

'Просроченный' AS status

FROM Carrier c

WHERE

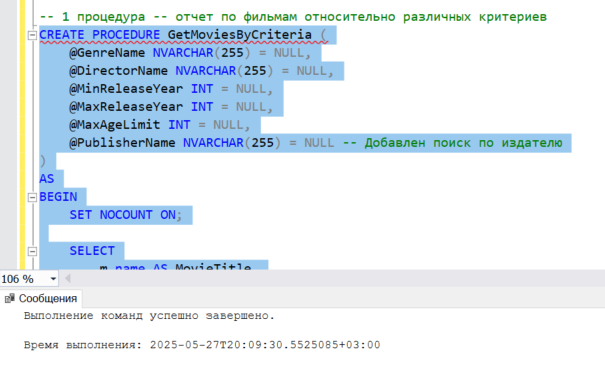
c.start\_rent IS NOT NULL -- он арендованный?

AND c.end\_rent < GETDATE() -- проверяем дату

ORDER BY c.end\_rent;

а теперь перейдем к лабораторной работе 14

1. Разработать 2 хранимые процедуры для формирования отчетных документов по специфике предметной области Вашего варианта индивидального задания и описанию выходных документов (ЛР-4).



CREATE PROCEDURE GetMoviesByCriteria (

@GenreName NVARCHAR(255) = NULL,

@DirectorName NVARCHAR(255) = NULL,

@MinReleaseYear INT = NULL,

@MaxReleaseYear INT = NULL,

@MaxAgeLimit INT = NULL,

@PublisherName NVARCHAR(255) = NULL -- Добавлен поиск по издателю

)

AS

BEGIN

SET NOCOUNT ON;

SELECT

m.name AS MovieTitle,

g.name AS Genre,

d.name AS Director,

p.name AS Publisher,

m.release\_year AS ReleaseYear,

m.duration AS DurationMinutes,

m.rating AS Rating,

m.age\_limit AS AgeLimit,

c.name AS Country

FROM Movie m

JOIN Genre g ON m.genre\_id = g.id

JOIN Director d ON m.director\_id = d.id

JOIN Publisher p ON m.publisher\_id = p.id

JOIN Country c ON m.country\_id = c.id

WHERE

(@GenreName IS NULL OR g.name = @GenreName)

AND (@DirectorName IS NULL OR d.name = @DirectorName)

AND (@MinReleaseYear IS NULL OR m.release\_year >= @MinReleaseYear)

AND (@MaxReleaseYear IS NULL OR m.release\_year <= @MaxReleaseYear)

AND (@MaxAgeLimit IS NULL OR m.age\_limit <= @MaxAgeLimit)

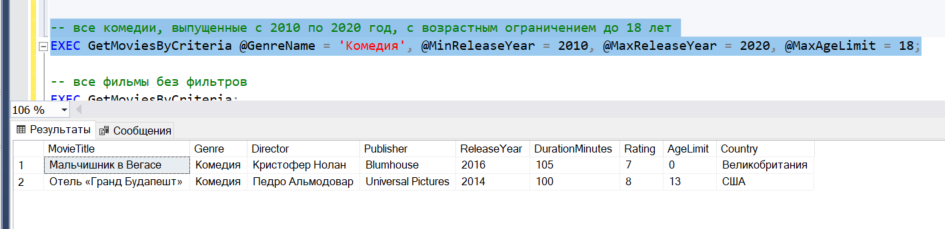
AND (@PublisherName IS NULL OR p.name = @PublisherName)

ORDER BY

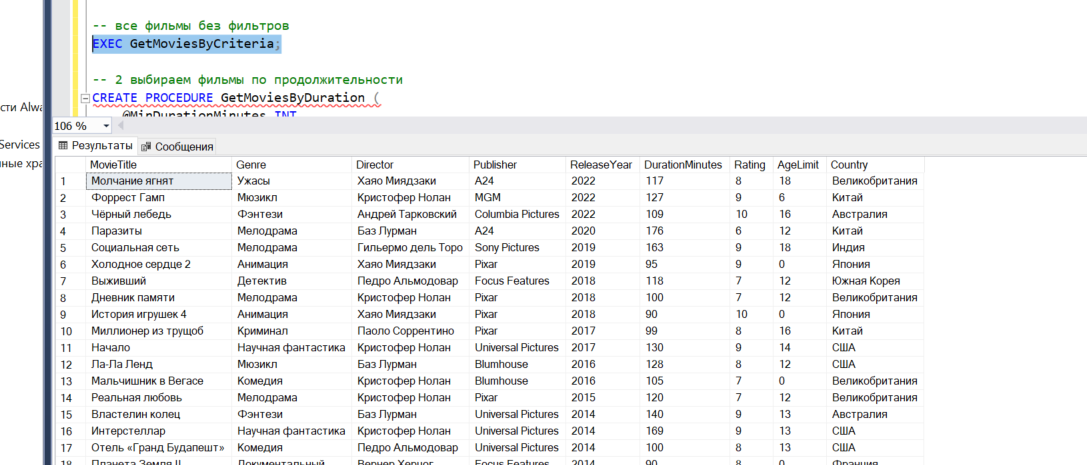
m.release\_year DESC, m.name;

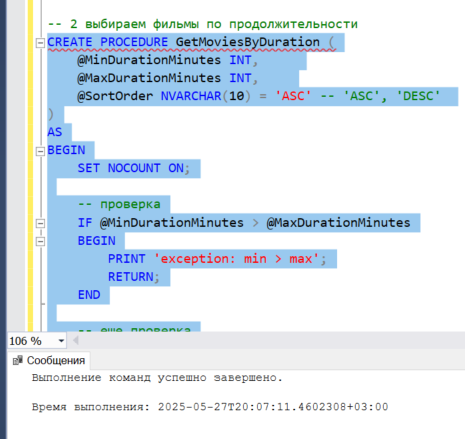
END

GO



EXEC GetMoviesByCriteria @GenreName = 'Комедия', @MinReleaseYear = 2010, @MaxReleaseYear = 2020, @MaxAgeLimit = 18;





CREATE PROCEDURE GetMoviesByDuration (

@MinDurationMinutes INT,

@MaxDurationMinutes INT,

@SortOrder NVARCHAR(10) = 'ASC' -- 'ASC', 'DESC'

)

AS

BEGIN

SET NOCOUNT ON;

-- проверка

IF @MinDurationMinutes > @MaxDurationMinutes

BEGIN

PRINT 'exception: min > max';

RETURN;

END

-- еще проверка

IF @SortOrder NOT IN ('ASC', 'DESC')

BEGIN

PRINT 'не ASC и не DESC, default = ASC';

SET @SortOrder = 'ASC';

END

SELECT

m.name AS MovieTitle,

m.duration AS DurationMinutes,

g.name AS Genre,

d.name AS Director,

m.release\_year AS ReleaseYear,

m.rating AS Rating,

m.age\_limit AS AgeLimit

FROM Movie m

LEFT JOIN Genre g ON m.genre\_id = g.id

LEFT JOIN Director d ON m.director\_id = d.id

WHERE

m.duration >= @MinDurationMinutes

AND m.duration <= @MaxDurationMinutes

ORDER BY

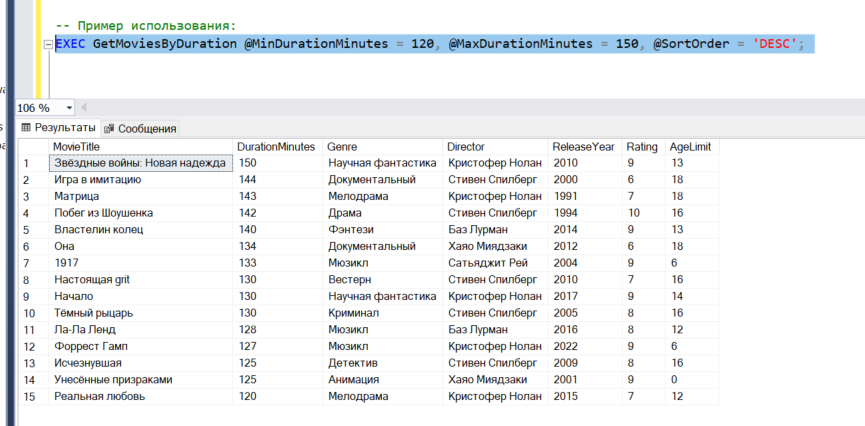
CASE WHEN @SortOrder = 'ASC' THEN m.duration END ASC,

CASE WHEN @SortOrder = 'DESC' THEN m.duration END DESC,

m.name ASC;

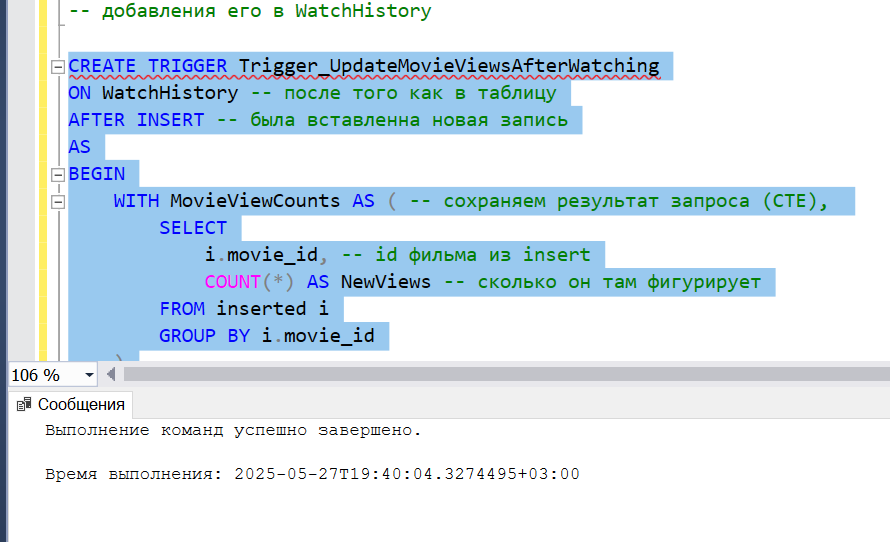
END

GO



EXEC GetMoviesByDuration @MinDurationMinutes = 120, @MaxDurationMinutes = 150, @SortOrder = 'DESC';

2. Разработать 1 триггер для поддержания ограничений целостности БД предметной области Вашего варианта индивидального задания или выполнения заданных действий с некоторыми ее таблицами после определенных DML-запросов пользователей.



CREATE TRIGGER Trigger\_UpdateMovieViewsAfterWatching

ON WatchHistory -- после того как в таблицу

AFTER INSERT -- была вставленна новая запись

AS

BEGIN

WITH MovieViewCounts AS ( -- сохраняем результат запроса (CTE),

SELECT

i.movie\_id, -- id фильма из insert

COUNT(\*) AS NewViews -- сколько он там фигурирует

FROM inserted i

GROUP BY i.movie\_id

)

UPDATE Movie -- и используем его для Обновления данных в Movie

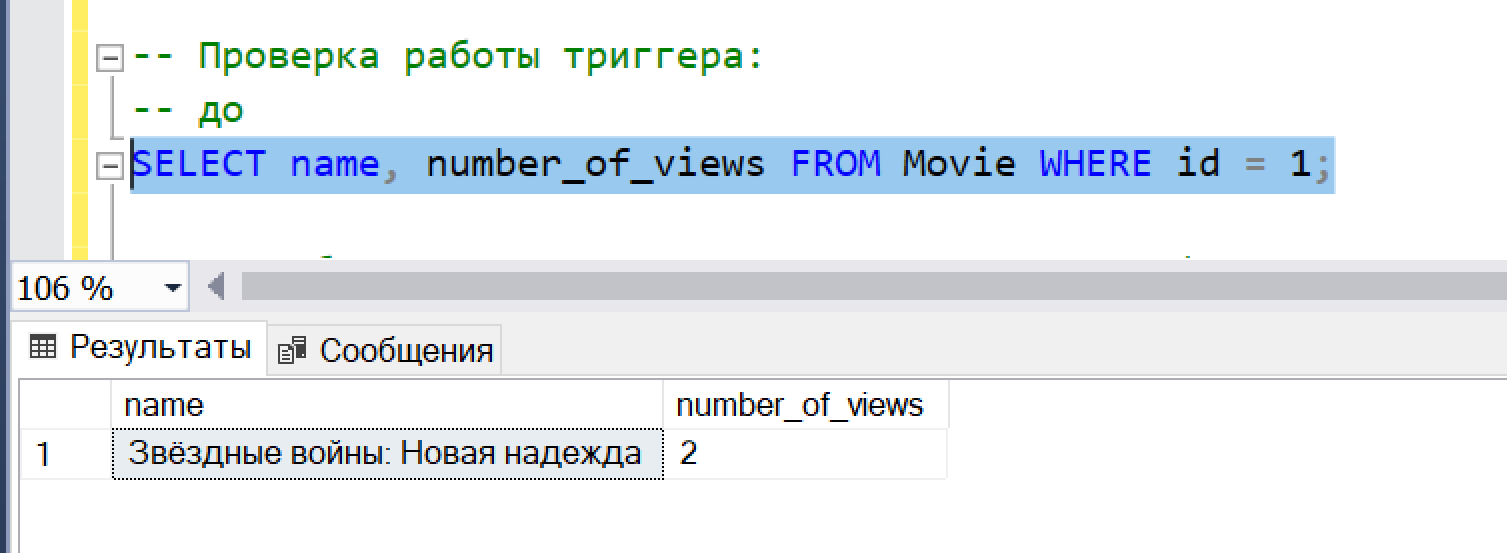
SET number\_of\_views = Movie.number\_of\_views + mvc.NewViews

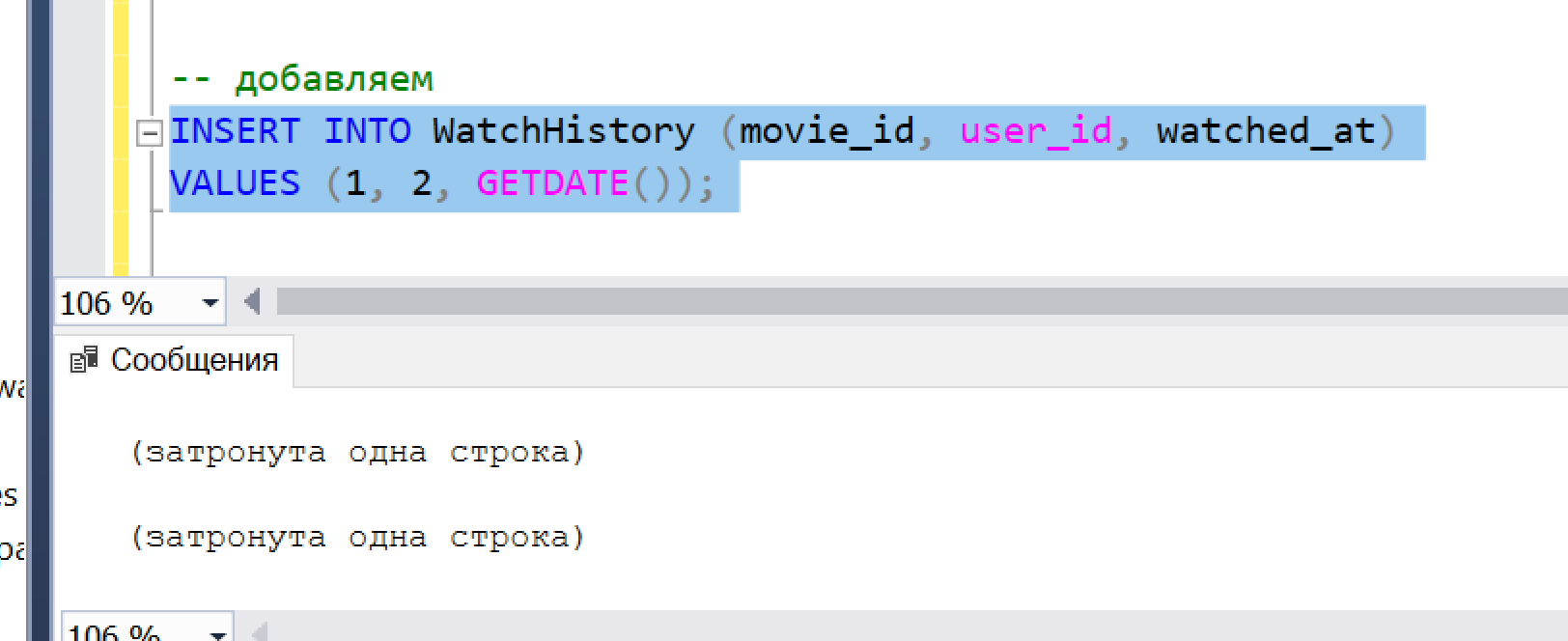
FROM Movie

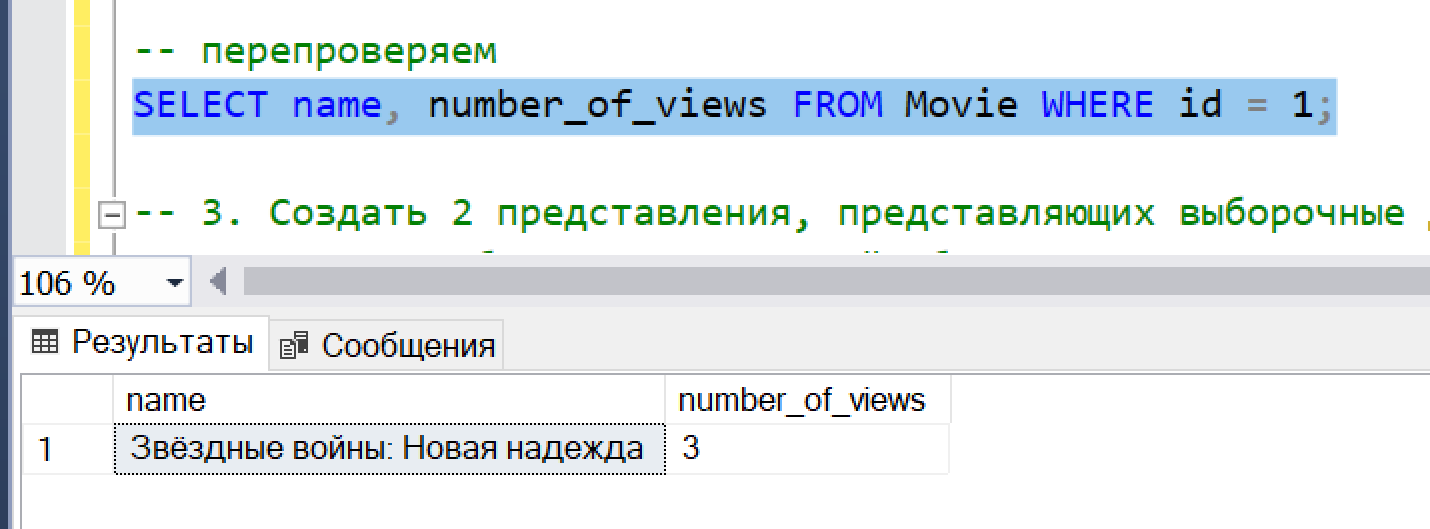
INNER JOIN MovieViewCounts mvc ON Movie.id = mvc.movie\_id;

END

GO







-- Проверка работы триггера:

-- до

SELECT name, number\_of\_views FROM Movie WHERE id = 1;

-- добавляем

INSERT INTO WatchHistory (movie\_id, user\_id, watched\_at)

VALUES (1, 2, GETDATE());

-- перепроверяем

SELECT name, number\_of\_views FROM Movie WHERE id = 1;

3. Создать 2 представления, представляющих выборочные данные отдельных таблиц БД предметной области вашего варианта индивидального задания.



CREATE VIEW view\_DetailedCarrierInfo

AS

SELECT

c.id AS Id, -- id

c.name AS Carrier\_Name, -- имя

t.name AS Carrier\_Type, -- тип

c.size AS Carrier\_SizeMB, -- размер

( -- фильмы через запятую

SELECT STRING\_AGG(CAST(m.name AS NVARCHAR(MAX)), ', ') WITHIN GROUP (ORDER BY m.name)

FROM MovieCarrier mc

JOIN Movie m ON mc.movie\_id = m.id

WHERE mc.carrier\_id = c.id

) AS MoviesOnCarrier,

c.start\_rent AS Start\_date\_,

c.end\_rent AS End\_date,

rc.name AS RentedFrom\_Name,

CASE -- cтатус носителя

WHEN c.rentaler\_concrete\_id IS NULL THEN 'Наше'

WHEN c.end\_rent IS NOT NULL AND c.end\_rent < GETDATE() THEN 'просрочил (до: ' + CONVERT(VARCHAR(10), c.end\_rent, 120) + ')'

WHEN c.end\_rent IS NOT NULL AND c.end\_rent >= GETDATE() THEN 'активно (до: ' + CONVERT(VARCHAR(10), c.end\_rent, 120) + ')'

WHEN c.start\_rent IS NOT NULL AND c.end\_rent IS NULL THEN 'не укащано когда вернуть'

END AS Carrier\_Status,

DATEDIFF(day, GETDATE(), c.end\_rent) AS Due

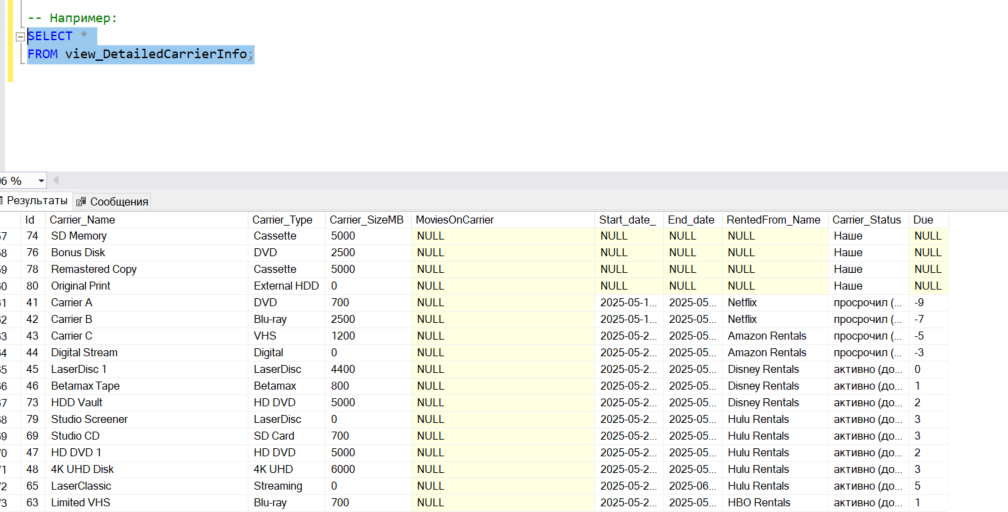
FROM Carrier c

JOIN Type t ON c.type\_id = t.id

LEFT JOIN Rentaler\_Concrete rc\_details ON c.rentaler\_concrete\_id = rc\_details.id

LEFT JOIN Rentaler rc ON rc\_details.rentaler\_id = rc.id

GO



SELECT \*

FROM view\_DetailedCarrierInfo;



CREATE VIEW view\_UserTopGenresOfMonth

AS

WITH MonthlyUserGenreViews AS (

SELECT

wh.user\_id,

g.id AS genre\_id,

g.name AS genre\_name,

COUNT(\*) AS amount\_of\_views,

ROW\_NUMBER() OVER (PARTITION BY wh.user\_id ORDER BY COUNT(\*) DESC, g.name ASC) as rang

FROM WatchHistory wh

JOIN Movie m ON wh.movie\_id = m.id

JOIN Genre g ON m.genre\_id = g.id

WHERE wh.watched\_at >= DATEADD(month, DATEDIFF(month, 0, GETDATE()), 0) -- левая рамка месяца

AND wh.watched\_at < DATEADD(month, DATEDIFF(month, 0, GETDATE()) + 1, 0) -- правая

AND wh.user\_id IS NOT NULL

GROUP BY wh.user\_id, g.id, g.name

)

SELECT

mu.user\_id,

u.name AS UserName,

mu.genre\_name AS GenreName,

mu.amount\_of\_views AS ViewsInMonth,

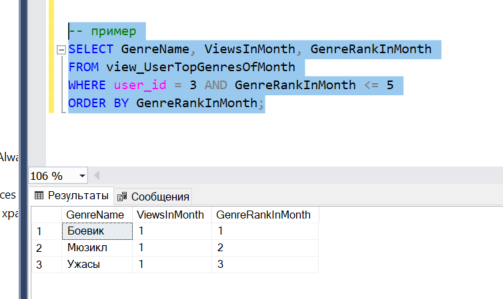
mu.rang AS GenreRankInMonth

FROM MonthlyUserGenreViews mu

JOIN [User] u ON mu.user\_id = u.id;

-- WHERE mu.rn <= 5;

GO



SELECT GenreName, ViewsInMonth, GenreRankInMonth

FROM view\_UserTopGenresOfMonth

WHERE user\_id = 3 AND GenreRankInMonth <= 5

ORDER BY GenreRankInMonth;