Database Homework 2
Database Design And SQL Programming

Sujith Madesh,309192

***Abstract*—In this second project of Databases course I designed a database of Movie_Catalog that stores 100's of movies.In the First three stages I designed the ER-diagram required for my task,created tables and filled in the tables with records matching the columns and then I created a few Index's that I thought would be useful in the later stages of the task where I had to write select queries as Indexes help selection by making its performance better.**
In the last two stages I write SELECT queries and Stored Procedure for automatic detection of Production delays along with them I prove if my written queries produce the desired output result.

I. Database Modelling

We create the first two tables of our database i.e [dbo].[Catalog_Movie] which is to have one to many relationships with [dbo].[Movie] .

```
CREATE TABLE [Catalog_Movie] (
   [catalog_ID] INTEGER NOT NULL,
   PRIMARY KEY ([catalog_ID])

);
```

```
CREATE TABLE [Movie] (
   [movie_ID] INTEGER NOT NULL,
   [category_ID] INTEGER not NULL,
   [catalog_ID] INTEGER not NULL FOREIGN KEY
       REFERENCES Catalog_Movie(catalog_ID),
   PRIMARY KEY ([movie_ID])
);
```

Next we will create dbo.[Movie_Characteristics] and dbo.[Title] we have created a [Title] table because as per the task the movie can have its title in two languages i.e in English and Original.We cam observe that [Movie_Characteristics] has one-to-one relationship with dbo.[Movie] and [Title] has one-to-one relationship with [Movie_Characteristics].In the [Movie_Characteristics] table I decided to have release_date, status_flag and delay_flag as Null because the movie might not be released yet.

```
CREATE TABLE [Movie_Characteristics] (
   [characteristics_ID] INTEGER not null,
   [production_startday] DATE not null,
   [expected_days] INTEGER not NULL,
   [release_date] DATE null,
   [description] VARCHAR(MAX) not null,
  [status_flag] BIT NULL ,
  [delay_flag] BIT NULL,
   [original_language] VARCHAR(100) not NULL,
   PRIMARY KEY ([characteristics_ID]),
  FOREIGN KEY ([characteristics_ID])
      REFERENCES MOVIE ([movie_ID]) );
```

```
CREATE TABLE [Title](
    [title] INTEGER,
   [title_original] varchar(255),
   [title_english] varchar(255),
   PRIMARY KEY ([title]),
  FOREIGN KEY ([title]) REFERENCES
      Movie_Characteristics
      ([characteristics_ID])
   );
```

Next we create dbo.[Categories] that stores the different genres of movie like action,adventure,sci-fi..etc and dbo.[Age_Category] that stores only three values green for anyone can watch,yellow for children can watch with adults supervision and red for adults.We can clearly say that [Movie] has one-to-many relationship with [Categories] and one-to-one relation with [Age_Category].

```
CREATE TABLE [Categories] (
   [category_ID] INTEGER not NULL,
   [movie_ID] INTEGER not NULL,
   [category_Name] VARCHAR(255) not NULL,
  Primary Key([category_ID]),
   FOREIGN KEY ([movie_ID]) REFERENCES Movie
       ([Movie_ID])
);
```

```
CREATE TABLE [Age_Category] (
   [age_category_ID] INTEGER not NULL,
   [restriction_color] VARCHAR(255) not NULL
  Primary key (age_category_ID)
  Foreign key (age_category_ID) References
      Movie (movie_ID)
  Constraint chk_restriction_color CHECK
      ([restriction_color] in
      ('green','yellow','red'))

);
```

Next we create four tables dbo.[Creator],dbo.[Actor],dbo.[Director] and dbo.[Job] I have assumed that anyone who works in the movie is Creator so Actor and Director both are creators but with different job titles or roles.So we can clearly say that [Creator] has one-to-one relationship with both [Actor] and [Director] whereas with [Job] it has one-to-many relationship because creator can have many functions in the movie.

```
CREATE TABLE [Creator] (
   [creator_ID] INTEGER NOT NULL IDENTITY(1,
```

```
      1),
   [name] VARCHAR(255) not NULL,
   [surname] VARCHAR(255) not NULL,
   [phone_number] VARCHAR(100) not NULL,
   [email] VARCHAR(255) not NULL,

   PRIMARY KEY ([creator_ID])
);
```

```
CREATE TABLE [Actor] (
   [ActorID] INTEGER not NULL IDENTITY(1, 1),
   [role] VARCHAR(255) NULL,
   [creator_ID] INTEGER not NULL,
  PRIMARY KEY ([creator_ID]),
  Foreign Key ([creator_ID]) references
      Creator ([creator_ID]),
  Constraint chk_restriction_role CHECK
      ([role] in ('actor','actress'))

);
```

```
CREATE TABLE [Director] (
   [DirectorID] INTEGER NOT NULL IDENTITY(1,
      1),
   [role] VARCHAR(255) NULL,
   [creator_ID] INTEGER not NULL,
   PRIMARY KEY ([creator_ID]),
  Foreign Key ([creator_ID]) references
      Creator ([creator_ID]),
  Constraint chk_restriction_role CHECK
      ([role] in ('director'))
);
```

```
Create Table [Job](
[creator_ID] int not null ,
[job_ID] int not null,
[function] varchar(200) null,
Primary key(job_ID),
Foreign key (creator_ID) references Creator
    (creator_ID)
);
```

As we know that in the production of movie there is more than one actor,creator or director so we will create the following tables dbo.[Movie_Cast] which has one-to-many relationship with [Actor] and [Movie] similarly for dbo.[Creator_Cast] with [Creator] and [Movie] and same goes for [Movie_Direction] with [Director] and [Movie].

```
Create Table [Movie_Cast](
[movie_ID] int not null ,
[actor_ID] int not null,
[character_name] VARCHAR(255) NULL,
Foreign key (actor_ID) references Actor
    (creator_ID),
Foreign key (movie_ID) references Movie
    (movie_ID)
);
```

```
Create Table [Creator_Cast](
[movie_ID] int not null ,
[creator_ID] int not null,
Foreign key (creator_ID) references Creator
    (creator_ID),
Foreign key (movie_ID) references Movie
    (movie_ID)

);
```

```
Create Table [Movie_Direction](
[movie_ID] int not null ,
[director_ID] int not null,
Foreign key (director_ID) references Director
    (creator_ID),
Foreign key (movie_ID) references Movie
    (movie_ID)
);
```

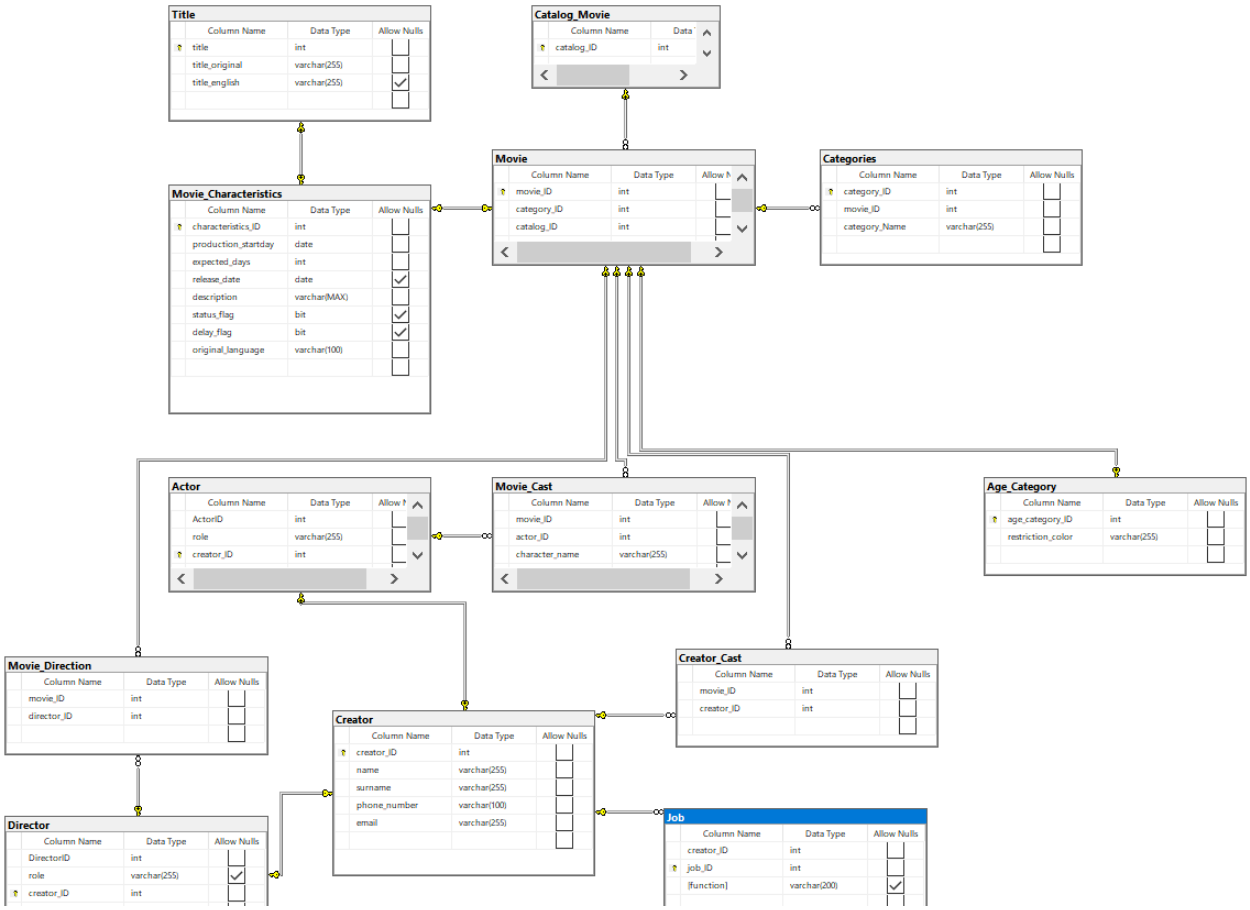


Fig. 1. ER-Diagram. The Model of my Movie Catalog

II. SQL Statements

I have inserted sample rows of 50 and 100 to each table in my database. Next, we have some DML operations to carry out and demonstrate the basic INSERT, UPDATE and DELETE statements. In order to be on the safe side and to avoid permanent changes in the data of the database we use begin transaction and rollback.

```
begin transaction
--Insert 1 row to our Movies table first then
    to the Movie_Characteristics
Insert into
    Movie(movie_ID,category_ID,catalog_ID)
    values(103,2,1);
Insert into
    Movie_Characteristics(characteristics_ID,
production_startday,expected_days,release_date,
description,status_flag,original_language)
    values(103,'2019-08-27',300,'2020-08-27'
,'This movie is super scary',1,'English');
Insert into Age_Category(age_category_ID,
restriction_color) values(103,'red');
Insert into
    Title(title,title_original,title_english)
    values (103,'Hitman','Hitman');
-----update--------------
update Movie_Characteristics set
    original_language='French' where
    characteristics_ID=103;
update Age_Category set
    restriction_color='green' where
    age_category_ID=103;

-----delete the created movie-----------
Delete from Movie_Characteristics where
    characteristics_ID=103
Delete from Age_Category where
    age_category_ID=103
Delete from Movie where movie_ID=103;
Delete from Title where title=103;


rollback transaction
```

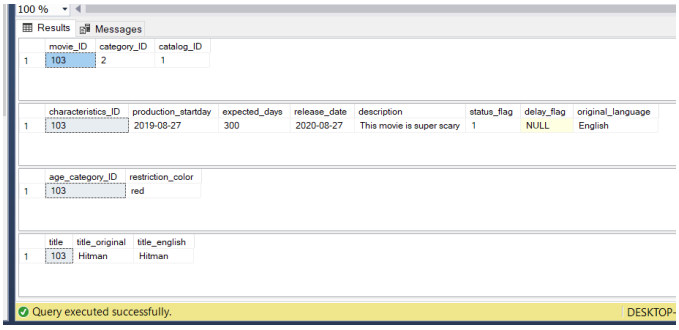


Fig. 2. Insert.

III. Designing INDEXS

In this part I created Non-Clustered Index mostly in order to speed up queries for specific reasons. As we know a table

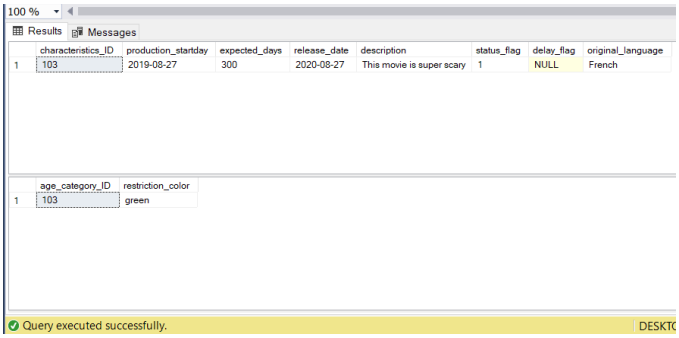


Fig. 3. Update.

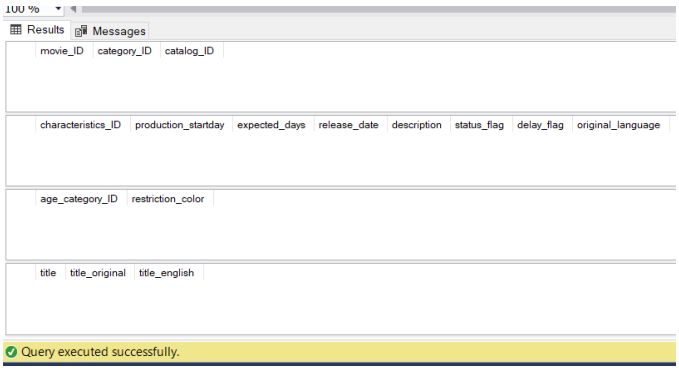


Fig. 4. Delete.

can have only one Clustered Index per table which is created during the creation of primary keys of that table, we leave it untouched so as not to overwrite any undesired information.

```
---Indexes--------------

--1.For query searches regarding Movie title
    on original title of the movie

Create Nonclustered Index
    [IDX_Title_title_original]
on Title (title_original ASC) INCLUDE
    (title_english);

--2.For query searches regarding movie of
    specific language

Create Nonclustered Index
    [IDX_Movie_Characteristics_original_language]
on Movie_Characteristics (original_language
    ASC) ;


--3.For query searches regarding latest movie
    releases and production date

Create Nonclustered Index
    [IDX_Movie_Characteristics_release_date]
on Movie_Characteristics (release_date
    Desc,production_startday Asc) ;

--4.For query searches related to movie
    categories
```

```
Create Nonclustered Index
    [IDX_Categories_category_name]
on Categories (category_name ASC);


--5.For query searches related to Age
    restriction of movie
Create Nonclustered Index
    [IDX_Age_Category_restriction_color]
on Age_Category (restriction_color ASC);


--6.For query searches related to Actor or
    Creator names acting in a movie

Create NonClustered Index [IDX_Creator_name]
on Creator ([name],surname ASC);


--7.For query searches related to Job of
    Creator in the movie
Create NonClustered Index [IDX_Job_function]
on Job ([function] ASC );

--8.For query searches related to
    character_name in a movie
Create NonClustered
    Index[IDX_Movie_Cast_character_name]
on Movie_Cast (character_name ASC);
```

There are more possible ways to create indexes, but we stop here for the sake of this Project. Just the disadvantage of non-clustered index is that it stores the columns in a different table with it's row locater to trace back to the original row of the specified table. In short, lookup process for such indexes become costly but on the other hand retrieving data becomes faster, we can avoid/reduce the overhead cost associated with clustered indexes.

IV. Selecting The Data

QUERY 1:

```
---4.1.Total number of movie releases by
    actors on a monthly window
with Monthly_Movie_release_by_actors as
(select YEAR(release_date) as
    movie_year,DATEPART(MONTH,release_date)
    as
    movie_month,CONCAT(Creator.name,Creator.surname)
    as actor_name,
count(*) over (partition by
    Creator.name,DATEPART(MONTH,release_date)
    ) as Monthly_movie_count from
    Movie_Characteristics
Join Movie on
    Movie.movie_ID=Movie_Characteristics.characteristics_ID
Join Movie_Cast on
    Movie_Cast.movie_ID=Movie.movie_ID
Join Actor on
    Actor.creator_ID=Movie_Cast.actor_ID
Join Creator on
    Creator.creator_ID=Actor.creator_ID)

select * from Monthly_Movie_release_by_actors
order by movie_year,movie_month
```

100 %

Results | Messages

	movie_year	movie_month	actor_name	Monthly_movie_count
1	2019	1	ActonWatkins	2
2	2019	1	ActonWatkins	2
3	2019	1	AidanWalker	1
4	2019	1	AureliaHiggins	2
5	2019	1	AureliaHiggins	2
6	2019	1	BuffyPeterson	1
7	2019	1	CarolMatthews	1
8	2019	1	ChandlerAyers	1
9	2019	1	ClareHoward	1
10	2019	1	CullenSchneider	1
11	2019	1	HeddaBooker	1
12	2019	1	JamesonValentine	1
13	2019	1	LeviDuffy	1
14	2019	1	NicholeSanford	1
15	2019	1	RashadSargent	1
16	2019	1	ScarletPhelps	1
17	2019	1	YaelBarrett	1
18	2019	1	YokoLambert	1
19	2019	8	BrennanStark	1
20	2019	9	ActonWatkins	1

Fig. 5. Select 4.1.

QUERY 2:

```
----4.2.A list of actors that can play in
    more than 2 languages
select name,surname , count(distinct
    original_language) as lang_count from
    Creator
join Actor on
    Actor.creator_ID=Creator.creator_ID
join Movie_Cast on
    Movie_Cast.actor_ID=Actor.creator_ID
join Movie on
    Movie_Cast.movie_ID=Movie.movie_ID
join Movie_Characteristics on
    Movie_Characteristics.characteristics_ID=Movie.movie_
GROUP BY name,surname
having COUNT(distinct original_language)>2;
```

To Prove 4.2: Let's take a actor and check which languages he has acted for example lets take actor name='Aidan'

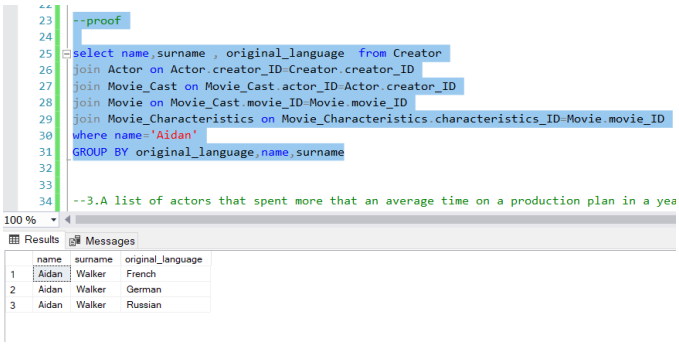
QUERY 3:

```
--4.3.A list of actors that spent more that
    an average time on a production plan in a
    yearly window
select name,surname
    ,avg(DATEDIFF(day,production_startday,
release_date)) as production_days from Creator
join Actor on
    Actor.creator_ID=Creator.creator_ID
join Movie_Cast on
    Movie_Cast.actor_ID=Actor.creator_ID
join Movie on
    Movie_Cast.movie_ID=Movie.movie_ID
join Movie_Characteristics on
    Movie_Characteristics.characteristics_ID=Movie.movie_
where
    DATEDIFF(YEAR,production_startday,release_date)<=1
group by name,surname,case when
```

	name	surname	lang_count
1	Chandler	Ayers	5
2	Yael	Barrett	4
3	Lewis	Black	3
4	Hedda	Booker	3
5	Timon	Burke	3
6	Simon	Hurley	3
7	Yoko	Lambert	3
8	Carol	Matthews	3
9	Scarlet	Phelps	3
10	Nichole	Sanford	3
11	Brennan	Stark	4
12	Aidan	Walker	3
13	Acton	Watkins	3

Query executed successfully.

Fig. 6. Select 4.2.



```
--proof

select name,surname , original_language  from Creator
join Actor on Actor.creator_ID=Creator.creator_ID
join Movie_Cast on Movie_Cast.actor_ID=Actor.creator_ID
join Movie on Movie_Cast.movie_ID=Movie.movie_ID
join Movie_Characteristics on Movie_Characteristics.characteristics_ID=Movie.movie_ID
where name='Aidan'
GROUP BY original_language,name,surname


--3.A list of actors that spent more that an average time on a production plan in a yea
```

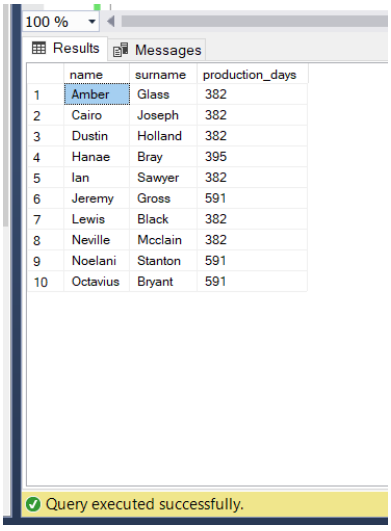
	name	surname	original_language
1	Aidan	Walker	French
2	Aidan	Walker	German
3	Aidan	Walker	Russian

Fig. 7. Proof 4.2.

```
    (DATEDIFF(YEAR,production_startday,release_date))<=1
    then 1 else 0 end
having
    avg(DATEDIFF(day,production_startday,release_date))
>(select
    AVG(DATEDIFF(day,production_startday,release_date))
    as total_avg from Movie_Characteristics
where DATEDIFF(YEAR,production_startday
,release_date)<=1);
```

Query 4:

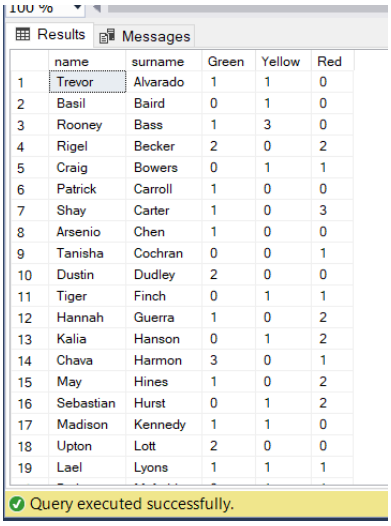
```
---4.4. A list of directors with the number
    of movies in each age category
select name,surname,
sum(Case when restriction_color='green' then
    1 else 0 end) as Green,
sum(Case when restriction_color='yellow' then
    1 else 0 end) as Yellow,
sum(Case when restriction_color='red' then 1
    else 0 end) as Red
from Creator
inner join Director on
    Director.creator_ID=Creator.creator_ID
inner join Movie_Direction on
    Movie_Direction.director_ID=Creator.creator_ID
inner join Movie on
    Movie.movie_ID=Movie_Direction.movie_ID
join Age_Category on
    Age_Category.age_category_ID=Movie.movie_ID
group by name,surname;
```



	name	surname	production_days
1	Amber	Glass	382
2	Cairo	Joseph	382
3	Dustin	Holland	382
4	Hanae	Bray	395
5	Ian	Sawyer	382
6	Jeremy	Gross	591
7	Lewis	Black	382
8	Neville	Mcclain	382
9	Noelani	Stanton	591
10	Octavius	Bryant	591

Query executed successfully.

Fig. 8. Select 4.3.



	name	surname	Green	Yellow	Red
1	Trevor	Alvarado	1	1	0
2	Basil	Baird	0	1	0
3	Rooney	Bass	1	3	0
4	Rigel	Becker	2	0	2
5	Craig	Bowers	0	1	1
6	Patrick	Carroll	1	0	0
7	Shay	Carter	1	0	3
8	Arsenio	Chen	1	0	0
9	Tanisha	Cochran	0	0	1
10	Dustin	Dudley	2	0	0
11	Tiger	Finch	0	1	1
12	Hannah	Guerra	1	0	2
13	Kalia	Hanson	0	1	2
14	Chava	Harmon	3	0	1
15	May	Hines	1	0	2
16	Sebastian	Hurst	0	1	2
17	Madison	Kennedy	1	1	0
18	Upton	Lott	2	0	0
19	Lael	Lyons	1	1	1

Query executed successfully.

Fig. 9. Select 4.4.

TO prove the query 4.4 we can show the number of movies directed by each director and compare it with result of the output of select.

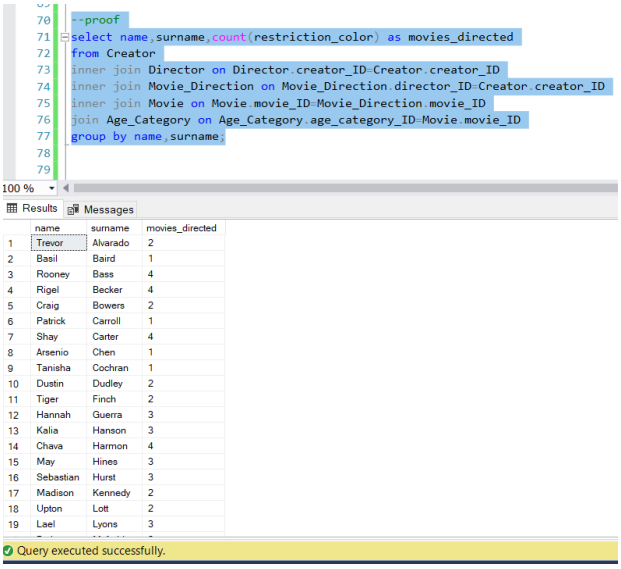


Fig. 10. Proof 4.4.

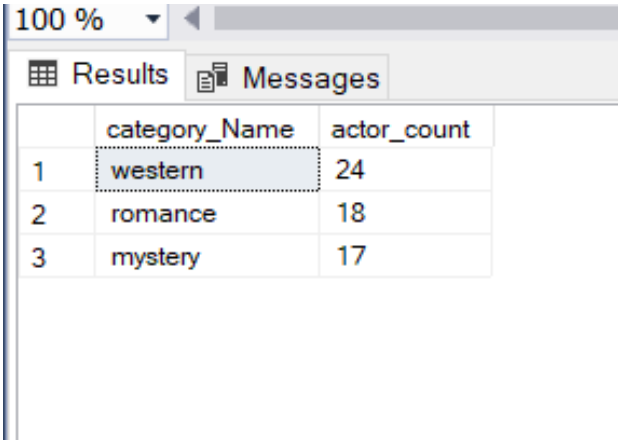


Fig. 11. Select 4.5

Query 5:

```
---4.5.Three Movie genres with highest number
    of actors
select top 3 category_Name ,Count( distinct
    Actor.creator_ID) as actor_count from
    Categories
join Movie on
    Movie.movie_ID=Categories.movie_ID
join Movie_Cast on
    Movie_Cast.movie_ID=Movie.movie_ID
join Actor on
    Actor.creator_ID=Movie_Cast.actor_ID
join Creator on
    Creator.creator_ID=Actor.creator_ID
group by category_Name
order by actor_count desc
```

To prove the query 4.5 we can check the number of actors in the western category as that has the highest number of actors.We see that it has 24 records and we can compare it

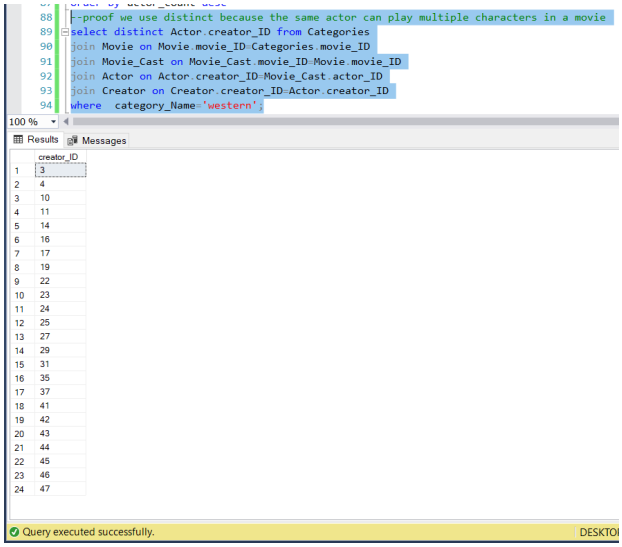


Fig. 12. Proof 4.5

to the output of 4.5

V. Stored Procedure

```
-------Stored Procedure------------
--Prepare a stored Procedure for automatic
    detection of production delays

Create Procedure SPproductionDelays
@production_time int
AS
declare @delayed_percent decimal(5,2)
declare @delayed_days float
declare @movie_ID int
declare @production_startday date
declare @release_date date
declare @movie_count int
declare @avg int
declare @delayedmovies_summary table(id
    int,dealyedDays float,durationPercent
    decimal(5,2),startday date,releasedate
    date,directorName
    varchar(50),directorSurname varchar(50))

declare delayed_movies cursor local for
select
    characteristics_ID,DATEDIFF(day,production_startday,
    as
    dealyed_days,production_startday,release_date,case
    when exists (select 1 from
    Movie_Characteristics)
then 1
else 0
end from Movie_Characteristics
where
    DATEDIFF(day,production_startday,release_date)>@prod
group by
    characteristics_ID,production_startday,release_date
open delayed_movies
fetch next from delayed_movies into
    @movie_ID,@delayed_days,@production_startday,@releas
```

```
IF @movie_count>0
BEGIN
WHILE @@FETCH_STATUS=0
BEGIN
----First we update the delay_flag of the
    movie

update Movie_Characteristics
set delay_flag=1 where
    characteristics_ID=@movie_ID;

--Assign an extra director to the movie that
    is not already present

Declare @flag int
Set @flag=1
Declare @director_ID int
Set @director_ID=(select top 1 creator_ID
    from Director order by NEWID())

While(@flag=1)
Begin
BEGIN
IF NOT EXISTS (SELECT * FROM Movie_Direction
WHERE director_ID=@director_ID

and movie_ID=@movie_ID)
BEGIN
INSERT INTO
    Movie_Direction(director_ID,movie_ID)
VALUES (@director_ID,@movie_ID)
PRINT 'The director_ID assigned is = ' +
    CONVERT(VARCHAR,@director_ID)

--Inserting director names and movie_id and
    delayed percent to table for calcualting
    avg worst %
set @delayed_percent=(cast (@production_time
    as float)/CAST(@delayed_days as float))
set @delayed_percent=(1-@delayed_percent)
set @delayed_percent=@delayed_percent*100
INSERT INTO
    @delayedmovies_summary(id,dealyedDays
,durationPercent,startday,releasedate,
directorName,directorSurname)
select @movie_ID,@delayed_days
,@delayed_percent,@production_startday
,@release_date,
Creator.name,Creator.surname from Creator
where Creator.creator_ID=@director_ID

Set @flag=0;
END
ELSE
Begin
Set @director_ID=(select top 1 creator_ID
    from Director order by NEWID())
End
END

end



fetch next from delayed_movies into
    @movie_ID,@delayed_days,
@production_startday
,@release_date,@movie_count

END
close delayed_movies
deallocate delayed_movies

---verifies which movie has worst duration %
    realtive to the average from last 3 years

set @avg=(select AVG(durationPercent) from
    @delayedmovies_summary)

select top 1 id as highest_delaymovie from
    @delayedmovies_summary
where startday>=DATEADD(YEAR,-3,GETDATE())
    and durationPercent>@avg
order by durationPercent desc



------Print Summary-------------
Print 'Summary of Delayed Movies'
select * from @delayedmovies_summary



END

ELSE
BEGIN
PRINT 'No Movies With Production Delays'
END
```

As per the task we have one input parameter @production_time which takes number of days it has taken for movie production. Firstly we select all movies which has taken more days in production than the @production_time given by the user then we set all those movies delay_flag=1 and assign a director that is not already working in that particular movie. Next I find the delayed percentage relative to the input value and insert it into a table along with names of the directors. From that we found the worst duration % movier relative to the average in the last 3 years.Finally I print the sumamary from the summary table where previously i had stored the values.The delay_flags for all movies is set to null before execution of this procedure. Let's Check how our store procedure will execute:

```
--checking:
declare @time int
set @time=450

begin transaction
exec SPproductionDelays @time

select * from Movie_Characteristics

rollback transaction

drop proc SPproductionDelays
```

100 %

Results | Messages

	highest_delaymovie
1	29

	id	dealyedDays	durationPercent	startday	releasedate	directorName	directorSurname
1	72	1575	71.00	2018-02-07	2022-06-01	Kalia	Hanson
2	79	1421	68.00	2018-06-28	2022-05-19	Trevor	Alvarado
3	29	1397	68.00	2018-06-13	2022-04-10	Craig	Bowers
4	66	1236	64.00	2018-11-17	2022-04-06	Tana	Sears
5	91	1161	61.00	2018-12-20	2022-02-23	Odysseus	Munoz
6	70	1371	67.00	2018-03-31	2021-12-31	Kalia	Hanson
7	63	1104	59.00	2018-12-18	2021-12-26	Shay	Carter
8	35	1317	66.00	2018-05-13	2021-12-20	Keith	Little
9	98	1340	66.00	2018-04-18	2021-12-18	Dustin	Dudley
10	3	1276	65.00	2018-06-13	2021-12-10	Keith	Little
11	55	1205	63.00	2018-07-22	2021-11-08	Jamal	Dickerson
12	95	1292	65.00	2018-04-21	2021-11-03	Candace	Stuart
13	90	1361	67.00	2018-02-07	2021-10-30	Curran	Villarreal
14	76	1194	62.00	2018-06-10	2021-09-16	Craig	Bowers

Fig. 13. Executing Stored Procedure

```
select * from Movie_Characteristics

rollback transaction

drop proc SPproductionDelays
```

100 %

Results | Messages

	characteristics_ID	production_startday	expected_days	release_date	description	status_flag	delay_flag	original_language
1	1	2018-02-22	235	2020-05-18	Fusce mollis. Duis sit amet diam eu dolor egestas rh...	1	1	Polish
2	2	2018-08-31	396	2021-03-15	luctus aliquet odio. Etiam ligula tortor, dictum eu, pla...	1	1	Portuguese
3	3	2018-06-13	242	2021-12-10	Praesent luctus. Curabitur egestas nunc sed libero. P...	1	1	Russian
4	4	2018-01-19	513	2020-03-05	lorem, sit amet ultricies sem magna nec quam. Cura...	1	1	Hindi
5	5	2018-10-06	38	2019-11-05	euismod mauris eu elit. Nulla facilisi. Sed neque. Se...	1	NULL	Russian
6	6	2018-10-25	326	2019-11-11	in consectetuer ipsum nunc id enim. Curabitur mass...	1	NULL	Chinese
7	7	2018-08-25	278	2021-08-19	a ultricies adipiscing, enim mi tempor lorem, eget m...	1	1	German
8	8	2018-08-05	697	2021-07-07	rhoncus id, mollis nec, cursus a, enim. Suspendisse ...	1	1	Arabic
9	9	2018-11-26	344	2019-01-15	iaculis quis, pede. Praesent eu dui. Cum sociis natoq...	1	NULL	French
10	10	2018-01-28	215	2019-09-11	porttitor scelerisque neque. Nullam nisl. Maecenas ...	1	1	English
11	11	2018-08-02	438	2021-09-02	semper rutrum. Fusce dolor quam, elementum at, e...	1	1	Kannada
12	12	2018-09-04	684	2020-11-17	ante ipsum primis in faucibus orci luctus et ultrices p...	1	1	French
13	13	2018-05-21	528	2021-05-29	et magnis dis parturient montes, nascetur ridiculus ...	1	1	Kannada
14	14	2018-03-25	310	2019-03-03	penatibus et magnis dis parturient montes, nascetur ...	1	NULL	English
15	15	2018-03-19	562	2021-04-12	Morbi sit amet massa. Quisque porttitor eros nec tell...	1	1	Hindi
16	16	2018-08-05	117	2019-11-20	In condimentum. Donec at arcu.	1	1	Dutch
17	17	2018-09-22	621	2021-03-28	justo. Proin non massa non ante bibendum ullamcor...	1	1	Kannada
18	18	2018-10-27	80	2020-01-09	ac ipsum. Phasellus vitae mauris sit amet lorem sem...	1	NULL	Hindi
19	19	2018-10-05	506	2021-03-15	rhoncus. Donec est. Nunc ullamcorper, velit in alique...	1	1	Hindi
20	20	2018-01-05	140	2019-06-03	scelerisque neque sed sem egestas blandit. Nam nu...	1	1	Japanese
21	21	2018-12-13	251	2019-08-27	a, auctor non, feugiat nec, diam. Duis mi enim, condi...	1	NULL	Chinese
22	22	2018-02-07	525	2020-06-02	ornare. Fusce mollis. Duis sit amet diam eu dolor eg...	1	1	German

Query executed successfully. DESKTOP-DGDGDUM (15.0 RTM)

Fig. 14. Checking delay flag

‘I certify that this assignment is entirely my own work, performed independently and without any help from sources which are not allowed’.

Sujith Madesh,309192