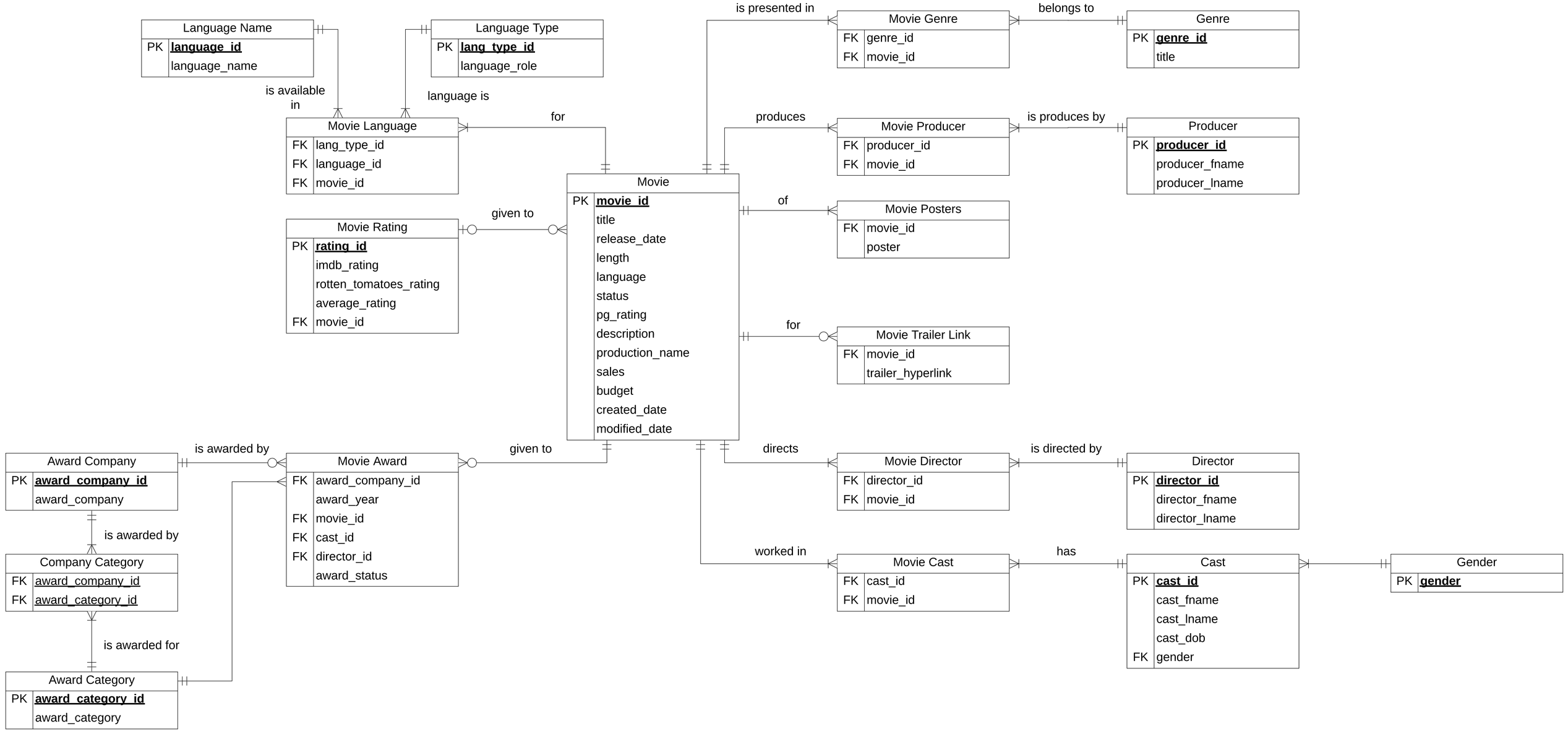
Movie Index Database Implementation

Abstract

With the rise of subscription platforms like Netflix and Prime, people have easy access to a huge database of movies and shows for their entertainment. We ease off the quantity available, people are now becoming picky about the movies and shows they would want to invest their time into. Delightful, instant experiences are at the heart of the best media and entertainment industry—and they require impeccable database performance. Media and entertainment companies looking to gain competitive advantage must create unique and dynamic digital experiences that drive both customer acquisition and retention. Thus, our database approach provides a robust and detailed approach to retrieve information swiftly for the end-users ensuring that their entertainment experience is not compromised.

Table of Contents:


Language Name
PK language_id
language_name
Language Type
PK lang_type_id
language_role
is available in
language is
Movie Language
FK lang_type_id
FK language_id
FK movie_id
for
Movie
PK movie_id
title
release_date
length
language
status
pg_rating
description
production_name
sales
budget
created_date
modified_date
is presented in
Movie Genre
FK genre_id
FK movie_id
belongs to
Genre
PK genre_id
title
produces
Movie Producer
FK producer_id
FK movie_id
is produces by
Producer
PK producer_id
producer_fname
producer_lname
given to
Movie Rating
PK rating_id
imdb_rating
rotten_tomatoes_rating
average_rating
FK movie_id
of
Movie Posters
FK movie_id
poster
for
Movie Trailer Link
FK movie_id
trailer_hyperlink
is awarded by
Award Company
PK award_company_id
award_company
given to
Movie Award
FK award_company_id
award_year
FK movie_id
FK cast_id
FK director_id
award_status
directs
Movie Director
FK director_id
FK movie_id
is directed by
Director
PK director_id
director_fname
director_lname
is awarded by
Company Category
FK award_company_id
FK award_category_id
is awarded for
Award Category
PK award_category_id
award_category
worked in
Movie Cast
FK cast_id
FK movie_id
has
Cast
PK cast_id
cast_fname
cast_lname
cast_dob
FK gender
Gender
PK gender

Stored Procedures

1. Movie Table Creation

```
CREATE PROCEDURE InsertMovie
        @title varchar(30),
        @release_date date,
        @running_time int,
        @language varchar(10),
        @status varchar(10),
        @pg_rating varchar(20),
        @description varchar(50),
        @production_name varchar(10),
        @sales double,
        @budget double
AS
BEGIN
        ---BEGIN TRANSACTION
                ---DECLARE @movie_id int;
                ---SELECT @movie_id = coalesce((select max(TagID) + 1 from movie_id), 1)
        ---COMMIT
        SET NOCOUNT ON;
      INSERT INTO


     Movie(title,release_date,length,[language],[status],pg_rating,description,production_name,sales,
     budget)

        VALUES(@title,@release_date,@running_time,@language,@status,@pg_rating,@description,
@production_name,
        @sales,@budget)
END
```

2. Top Rating movies based on Genre

```
create procedure [dbo].[Top_Movies] @genre Char(10)
as
begin
select top (10) m.title, m.description, g.title ,r.average_rating
from Movie m
```

```
join MovieRating r on r.movie_id = m.movie_id
join MovieGenre mg on mg.movie_id = m.movie_id
join Genre g on mg.genre_id = g.genre_id
where g.title = @genre
order by r.average_rating desc
end
GO
exec [Top_Movies] @genre = 'Thriller'
```

3. Total Movies Based For AGenre

```
create procedure CountMoviesOnGenre @genre varchar(20), @countgenre int out
as
begin
select count(*) CountOfGenres
from Movie m
        inner join MovieGenre mg on
                m.movie_id = mg.movie_id
                inner join Genre g on
                        mg.genre_id = g.genre_id
where g.title = @genre
group by mg.genre_id

select @countgenre = @@ROWCOUNT
end
```

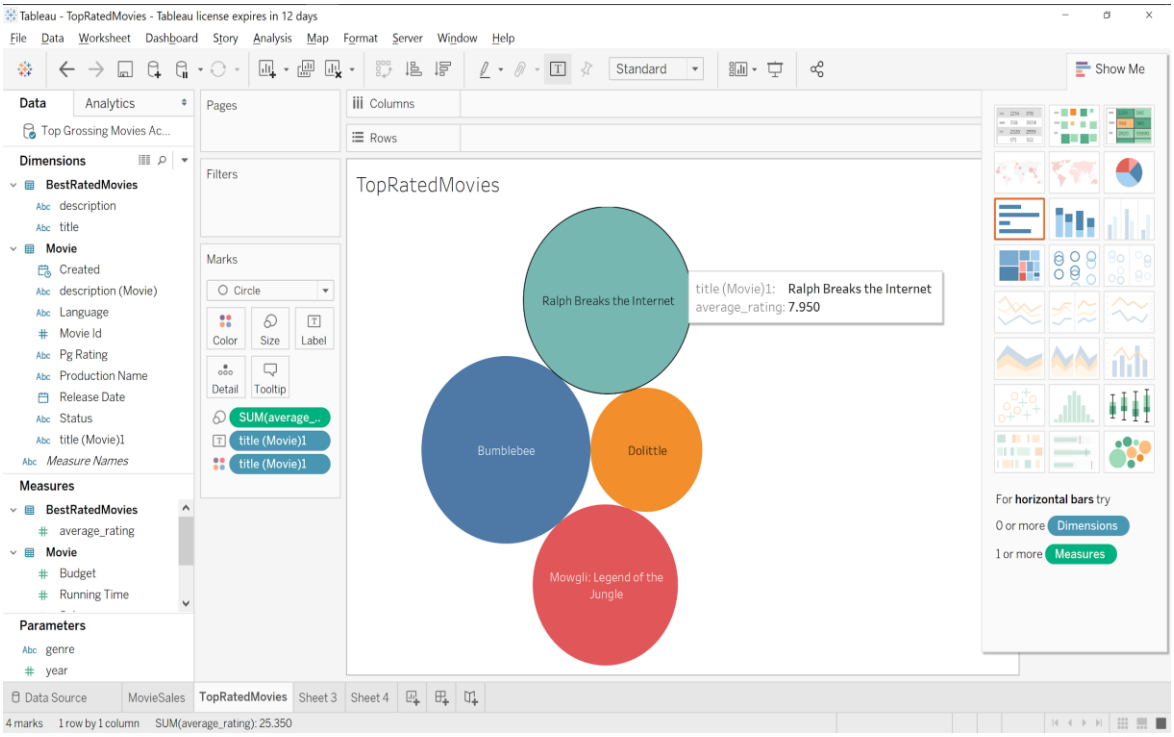
Command to execute:

```
declare @countgenre int
exec CountMoviesOnGenre 'Adventure' , @countgenre OUT
```

Views:

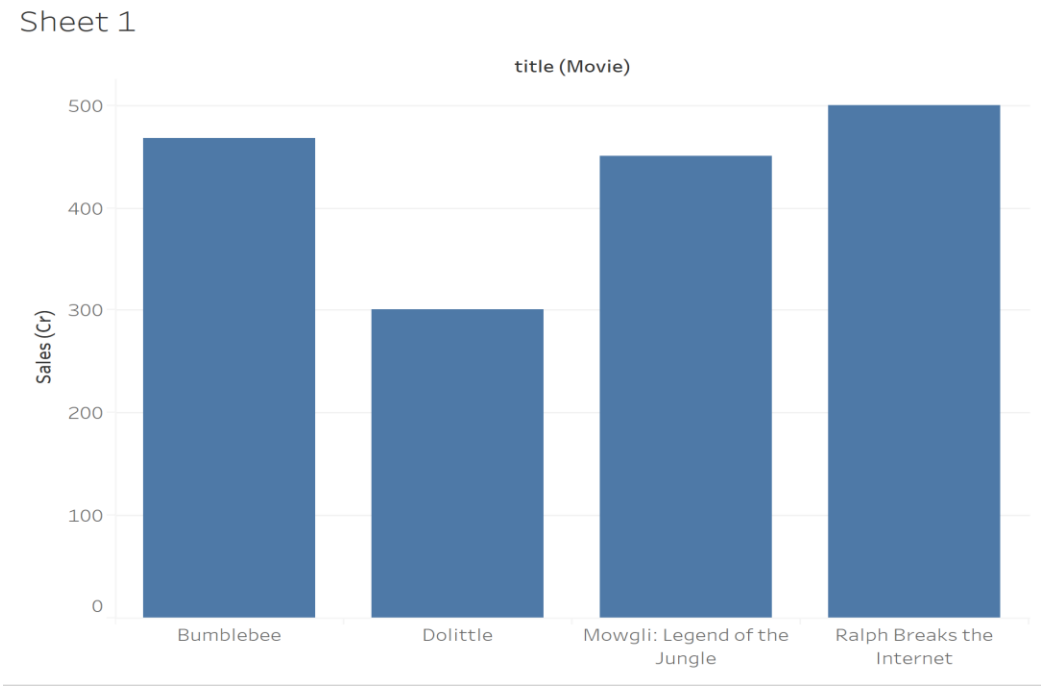
1. Top Rated Movies

Create view BestRatedMovies as
select top(4) Movie.title, Movie.description, MovieRating.average_rating
from Movie inner join MovieRating
on Movie.movie_id=MovieRating.movie_id
order by MovieRating.average_rating desc

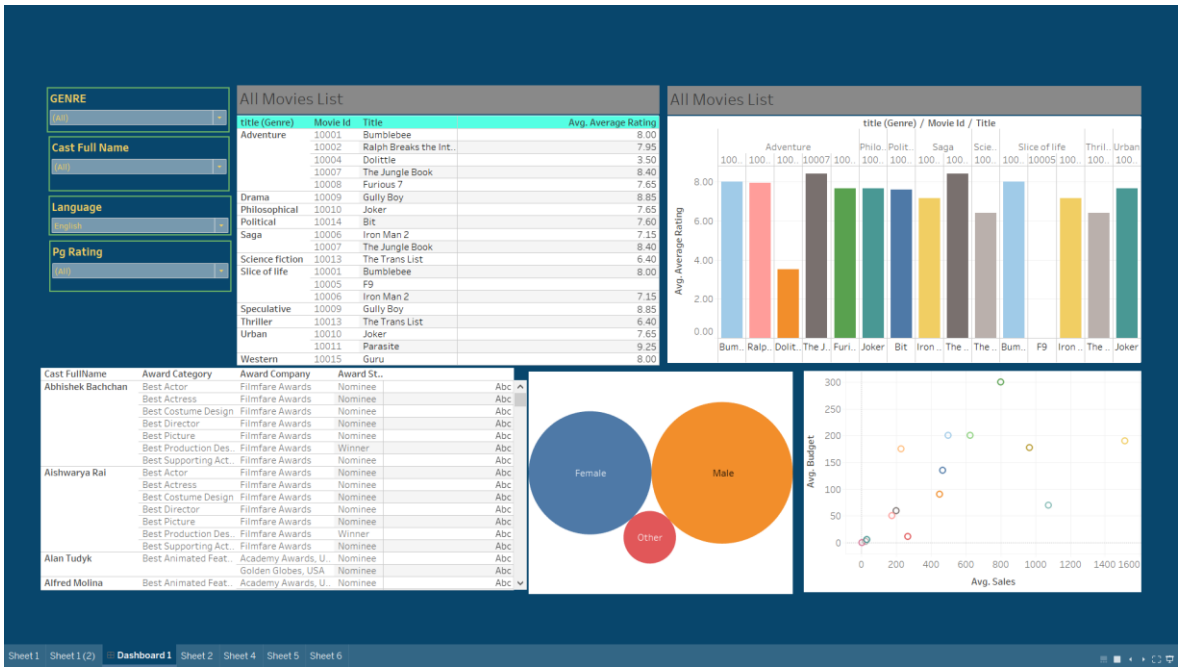


2. Top Grossing Movie for a year

```
select top(3) title
from Movie
where (SELECT YEAR(release_date)) = '2018'
order by sales desc
```



3. Tableau Dashboard



Trigger:

1. Trigger for modified date

```
CREATE TRIGGER trg_UpdateTimeEntry
ON Movie
AFTER UPDATE
AS
UPDATE Movie
SET modified_date = GETDATE()
WHERE movie_id IN (SELECT DISTINCT movie_id FROM Inserted);
```

Event trigger :

```
CREATE EVENT updaterelease_status
   ON SCHEDULE
     EVERY 12 HOUR
       DO
update Movie
set status='Released'
where release_date < GETDATE()
update Movie
set status='Not Released'
where release_date > GETDATE();
```

Table-level CHECK Constraints:

1. **Check Constraint in Movie Table creation to ensure running_time, budget and sales are not 0.**

```
CONSTRAINT CHK_running_time CHECK (running_time>0),
CONSTRAINT CHK_budget CHECK (budget>0),
CONSTRAINT CHK_sales CHECK (sales>=0)
```

2. **Check Constraint in MovieAward Table creation to check the AwardStatus**

```
CREATE TABLE MovieAward (
award_company_id INT FOREIGN KEY REFERENCES AwardCompany(award_company_id),
award_category_id INT FOREIGN KEY REFERENCES AwardCategory(award_category_id),
cast_id INT FOREIGN KEY REFERENCES Cast(cast_id),
award_year INT NOT NULL,
award_status char(10) ,
CHECK (award_status IN ('Winner', 'Nominee')),
movie_id int FOREIGN KEY REFERENCES Movie(movie_id),
director_id int FOREIGN KEY REFERENCES Director(director_id)
);
```

Computed Columns based on a function:

1. Average_rating from Rating table

```
CREATE TABLE MovieRating (
rating_id int NOT NULL PRIMARY KEY IDENTITY(80001,1),
imdb_rating float,
rottentomatoes_rating float,
average_rating as (imdb_rating+rottentomatoes_rating)/2,
movie_id int FOREIGN KEY REFERENCES Movie(movie_id)
);
```

	rating_id	imdb_rating	rottentomatoes_rating	average_rating	movie_id
1	80001	6.8	9.2	8	10001
2	80002	7.1	8.8	7.95	10002
3	80003	6.5	5.3	5.9	10003
4	80004	5.5	1.5	3.5	10004
5	80005	NULL	NULL	NULL	10005

Column Data Encryption:

/*Encryption*/

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Qwerty@123';

ALTER TABLE Cast$

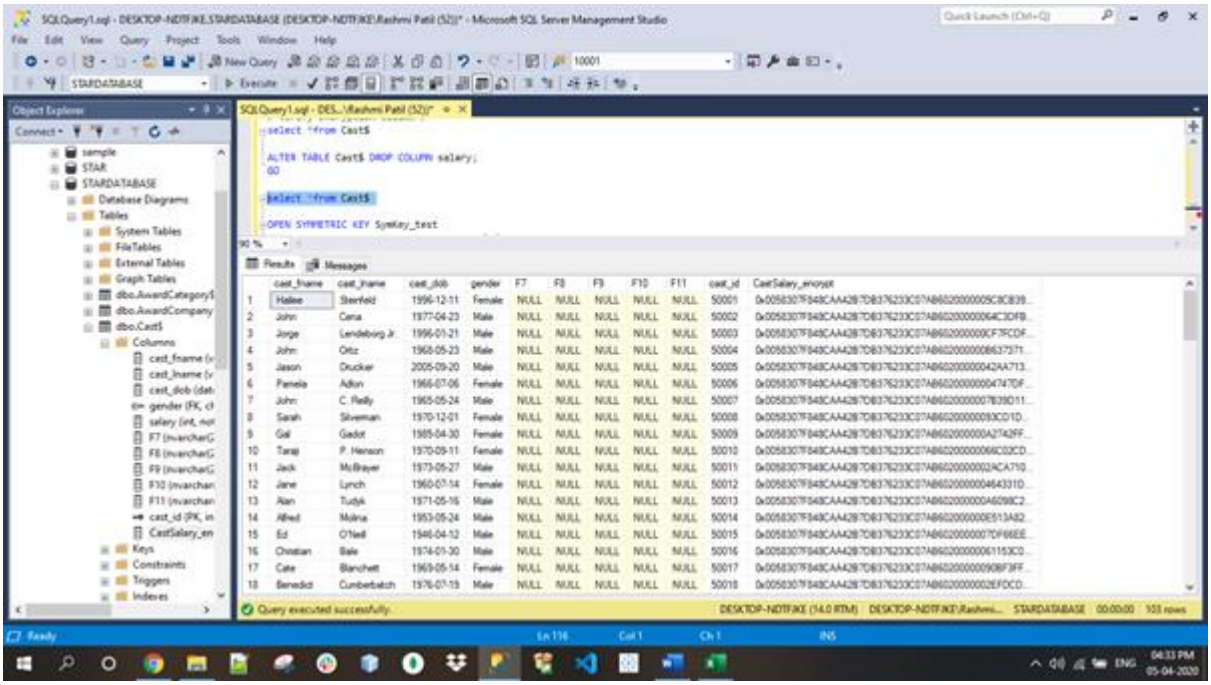
ADD CastSalary_encrypt varbinary(MAX)

UPDATE Cast$

SET CastSalary_encrypt = ENCRYPTBYKEY (KEY_GUID ('SymKey_test'), cast_lname)

From Cast$

GO



/*Verify Encryption column*/

select *from Cast$

/*Delete old column*/

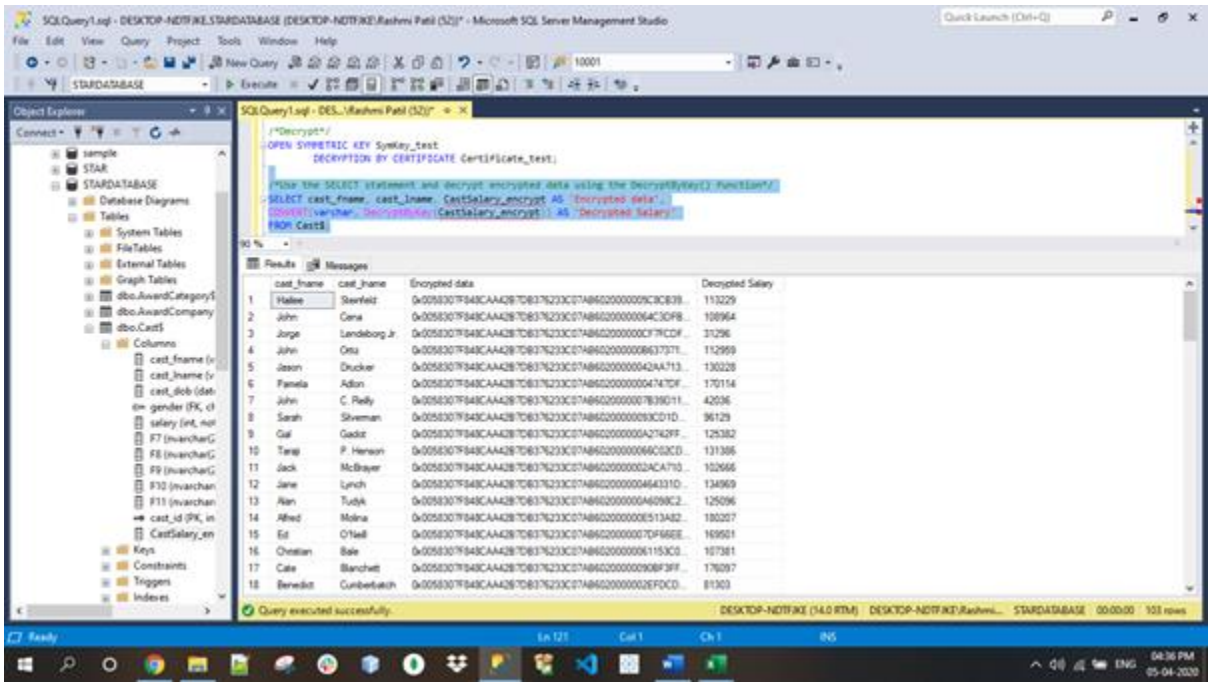
ALTER TABLE Cast$ DROP COLUMN salary;

GO

/*Decrypt*/

/*Use the SELECT statement and decrypt encrypted data using the DecryptByKey() function*/

SELECT cast_fname, CastSalary_encrypt AS 'Encrypted data',
CONVERT(varchar, DecryptByKey(CastSalary_encrypt)) AS 'Decrypted Salary'
FROM Cast$;



Non-clustered indexes

1. **A NonClustered index for the Cast table.**
   We have created index IX_Cast_Title that stores all the last names of the Cast.
   Create NonClustered index IX_Cast_Title on Cast$ (cast_lname ASC)

   The following query is executed on the created index which ensures faster query execution
   SELECT cast_id, cast_fname, cast_lname
   FROM cast$
   WHERE cast_lname = 'Angel'