TERM PROJECT _ FINAL

Online DVD Rental Business

FARIBORZ NOROUZI

SPRING 1 _ 2017

Contents: Page

Objective:

Create an initial design for the database schema for an online DVD rental business that is similar to the DVD rental portion of the business pioneered by *NetFlix*®.
I mapped my conceptual design into my approved DBMS, and implemented the database schema.

Business Rules:

Business Rules are used to define entities, attributes, relationships and constraints. Usually though they are used for the organization that stores or uses data to be an explanation of a policy, procedure, or principle. For this particular project following business rules are defined.

- **Related between CUSTOMER and ADDRESS entities**

✓ A Customer must have an address. (Mandatory – Singularity)
✓ Each Address must be corresponded to one or more Customers. (Mandatory – Plurality)

- **Related between CUSTOMER and MEMBERSHIP entities**

✓ A Customer can enroll zero to many Membership plan. (Optionality – Plurality)
✓ Membership plan can be enrolled by zero to many Customers. (Optionality – Plurality)

- **Related between CUSTOMER and QUEUE_MOVIE entities**

✓ Each Customer has zero or many Queue_movie. (Optionality – Plurality)
✓ A Queue_Movie must be had by one Customer. (Mandatory – Singularity)

- **Related between CUSTOMER and RENTAL entities**

✓ A Customer has zero or many Rental. (Optionality – Plurality)

✓ Each Rental must be had by one Customer. (Mandatory – Singularity)

- **Related between CUSTOMER and PAYMENT entities**

✓ A Customer can have zero to many Payment. (Optionality – Plurality)

✓ Each Payment must be related to one Customer. (Mandatory – Singularity)

- **Related between MOVIE and QUEUE_MOVIE entities**

✓ Each Movie can correspond to zero or many Queue_Movie. (Optionality – Plurality)

✓ Each Queue_Movie must be corresponded by one Movie. (Mandatory – Singularity)

- **Related between MOVIE and RENTAL entities**

✓ Each Movie can correspond to zero or many Rental. (Optionality – Plurality)

✓ Each Rental must be corresponded by one Movie. (Mandatory – Singularity)

- **Related between DIRECTOR and MOVIE entities**

✓ Each Director can direct one to many Movies. (Optionality – Plurality)

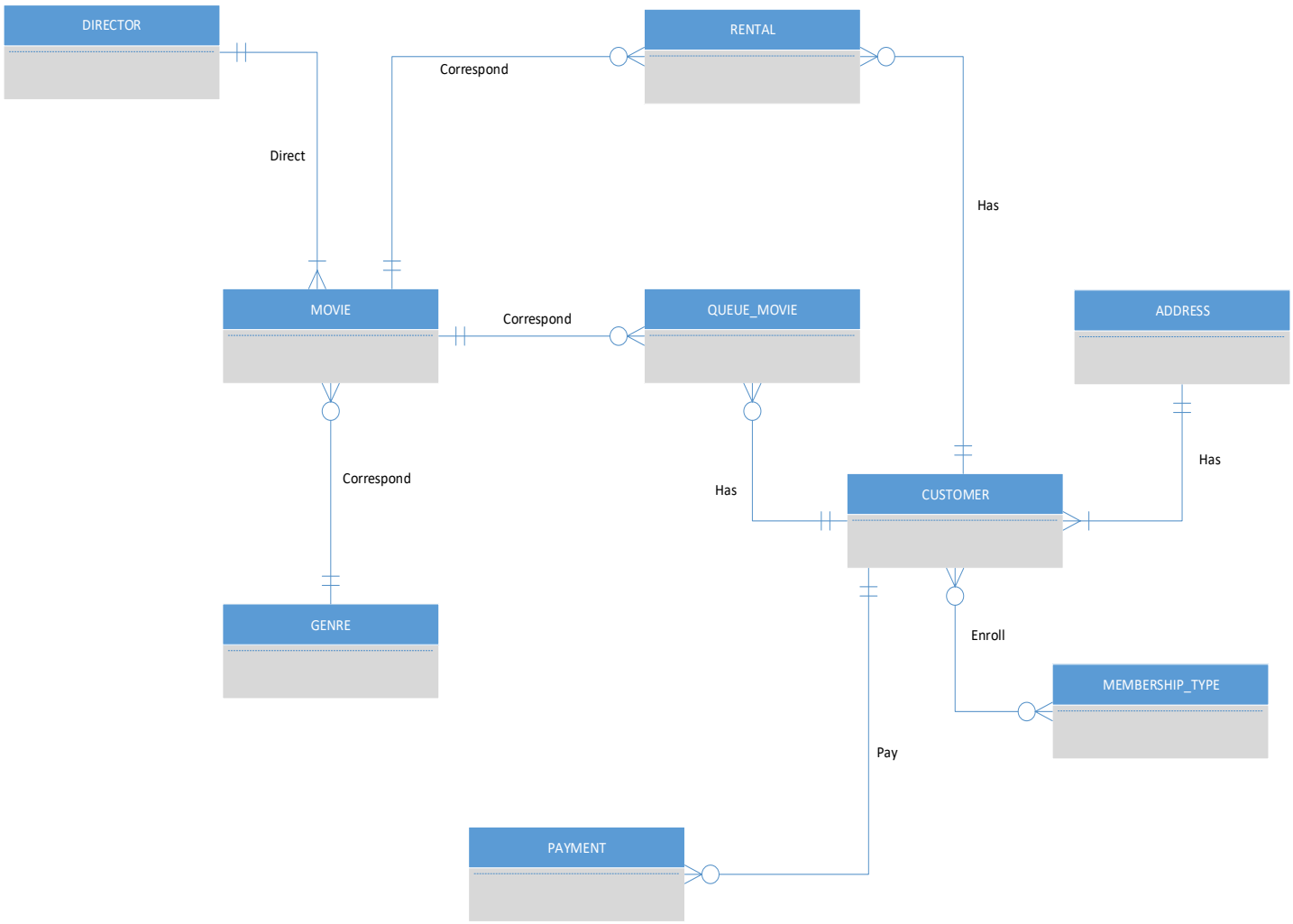
✓ Any Movie must be directed by a Director. (Mandatory – Singularity)

- **Related between MOVIE and GENRE entities**

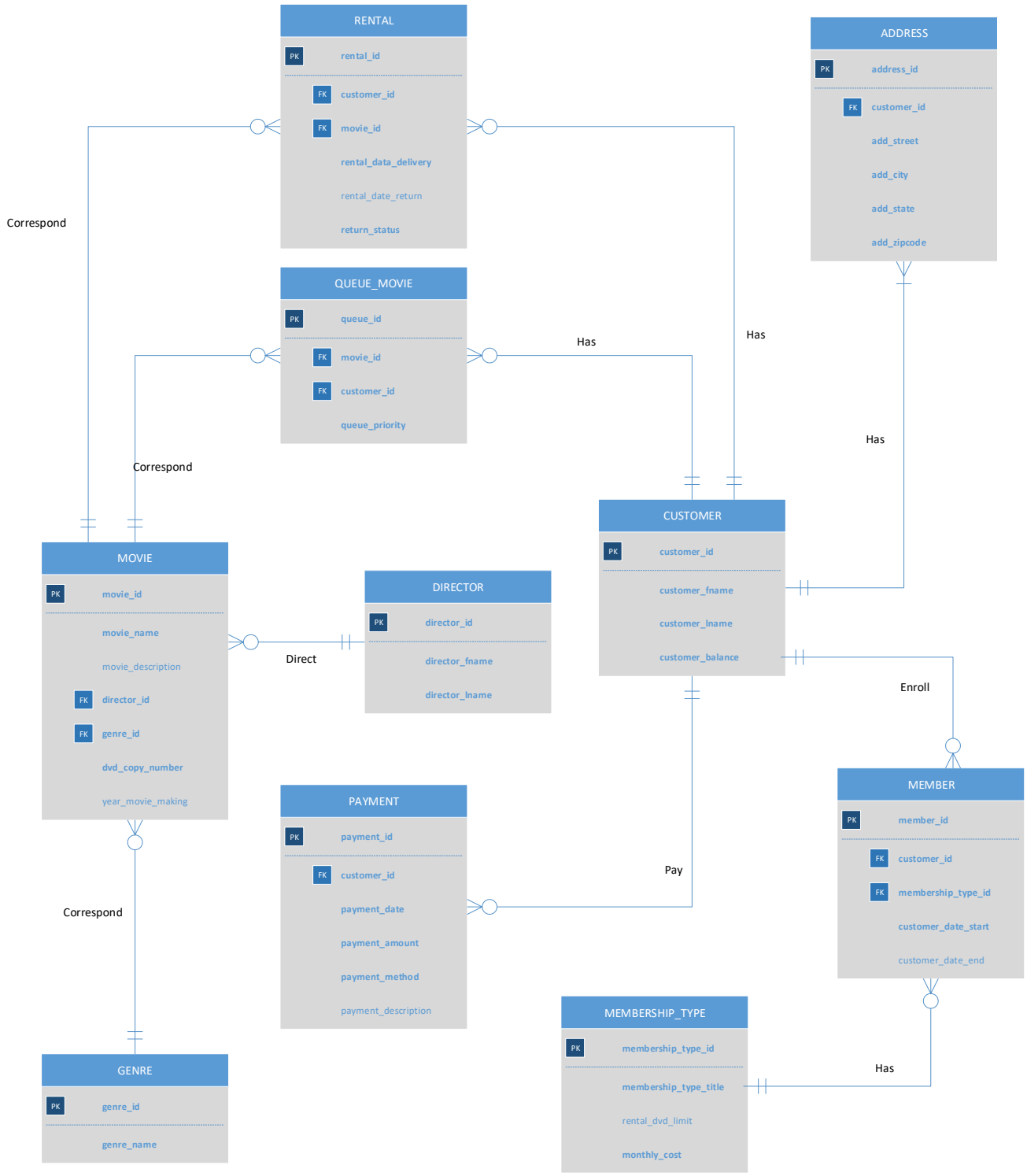
✓ Each movie must correspond to one Genre. (Mandatory – Singularity)

✓ A Genre can be corresponded zero to many Movies. (Optionality – Plurality)

Conceptual ERDs:



Logical ERDs:



Use Case 1 – Rental:

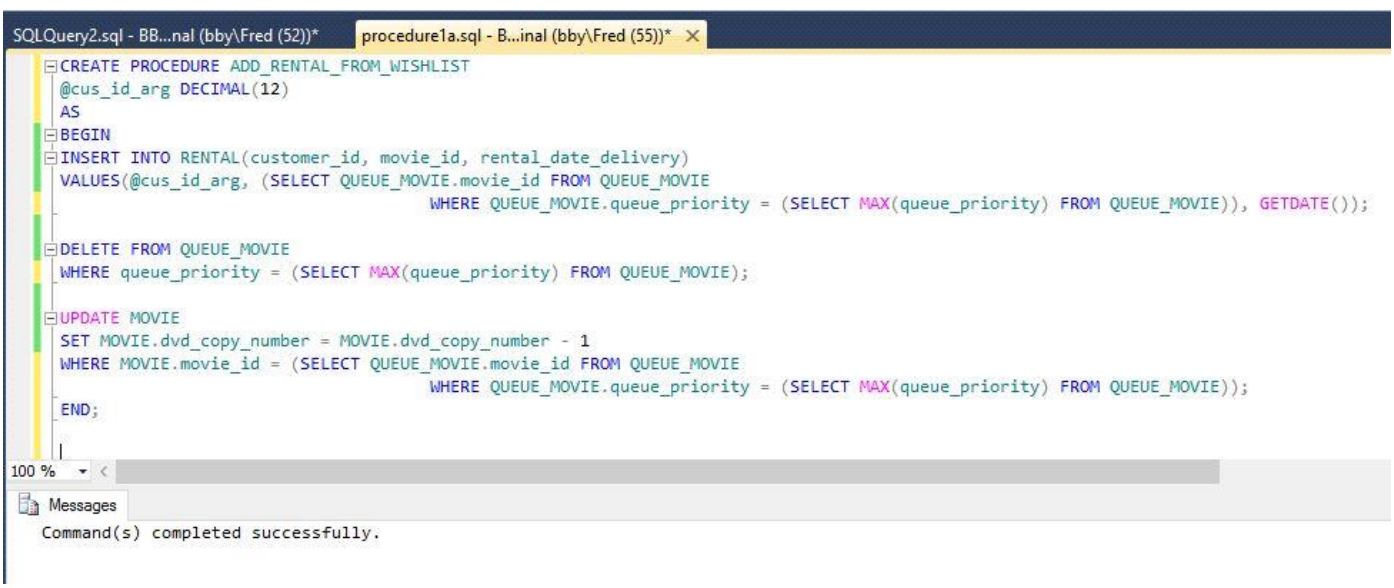
a. A customer rents the movies "X-Men: Days of Future Past" and "The Angry Birds Movie". Develop a parameterized stored procedure that supports a customer renting a movie, then invoke the stored procedure two times (to satisfy the use case) for a customer of your choosing.

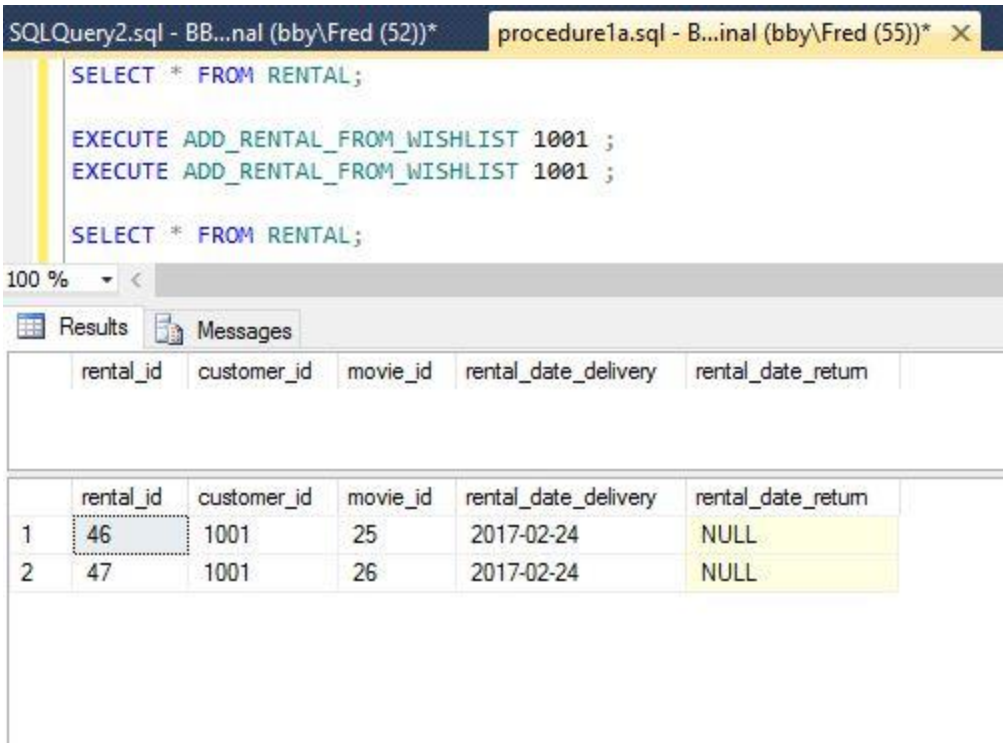
Code:

```
CREATE PROCEDURE ADD_RENTAL_FROM_WISHLIST
@cus_id_arg DECIMAL(12)
AS
BEGIN
INSERT INTO RENTAL(customer_id, movie_id, rental_date_delivery)
VALUES(@cus_id_arg, (SELECT QUEUE_MOVIE.movie_id FROM QUEUE_MOVIE
                     WHERE QUEUE_MOVIE.queue_priority = (SELECT MAX(queue_priority)
                     FROM QUEUE_MOVIE)), GETDATE());

DELETE FROM QUEUE_MOVIE
WHERE queue_priority = (SELECT MAX(queue_priority) FROM QUEUE_MOVIE);

UPDATE MOVIE
SET MOVIE.dvd_copy_number = MOVIE.dvd_copy_number - 1
WHERE MOVIE.movie_id = (SELECT QUEUE_MOVIE.movie_id FROM QUEUE_MOVIE
                        WHERE QUEUE_MOVIE.queue_priority = (SELECT MAX(queue_priority)
FROM QUEUE_MOVIE));
END;
```





b. A customer requests the titles of all movies they have rented that are directed by "George Lucas" and "Rich Christiano". Write a single query that retrieves this information for a customer of your choosing.

Code:

```
SELECT DIRECTOR.director_fname, DIRECTOR.director_lname, MOVIE.movie_name
FROM RENTAL
JOIN MOVIE ON RENTAL.movie_id  = MOVIE.movie_id
AND RENTAL.customer_id = 1001
JOIN DIRECTOR ON MOVIE.director_id = DIRECTOR.director_id
WHERE DIRECTOR.director_fname = 'George' AND DIRECTOR.director_lname = 'Lucas'
 OR DIRECTOR.director_fname = 'Rich' AND DIRECTOR.director_lname = 'Christiano' ;
```

SQLQuery1.sql - BB...nal (bby\Fred (55))* | procedure1b.sql - B...ster (bby\Fred (52))*

```
SELECT DIRECTOR.director_fname, DIRECTOR.director_lname, MOVIE.movie_name
FROM RENTAL
JOIN MOVIE ON RENTAL.movie_id  = MOVIE.movie_id
AND RENTAL.customer_id = 1001
JOIN DIRECTOR ON MOVIE.director_id = DIRECTOR.director_id
WHERE DIRECTOR.director_fname = 'George' AND DIRECTOR.director_lname = 'Lucas'
 OR DIRECTOR.director_fname = 'Rich' AND DIRECTOR.director_lname = 'Christiano' ;
```

100 %

Results | Messages

	director_fname	director_lname	movie_name
1	George	Lucas	Star Wars
2	George	Lucas	The Empire Stricks Back
3	George	Lucas	Return the Jedi
4	George	Lucas	Star Wars: The Force Awakense
5	Rich	Christiano	Time Changer
6	Rich	Christiano	A Matter of Faith

Use Case 2 - Movie Queue:

a. A customer adds a movie to their queue so that the newly added movie will be the next movie they receive. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure for a customer and movie of your choosing.

Code:

```
CREATE PROCEDURE ADD_MOVIE_QUEUE
@mov_id_arg DECIMAL(12),
@cus_id_arg DECIMAL(12)
AS
BEGIN
UPDATE QUEUE_MOVIE
SET QUEUE_MOVIE.queue_priority = QUEUE_MOVIE.queue_priority + 1
WHERE customer_id = @cus_id_arg;

INSERT INTO QUEUE_MOVIE(movie_id, customer_id, queue_priority)
VALUES(@mov_id_arg, @cus_id_arg, 1);
END;
```

SQLQuery3.sql - BB...nal (bby\Fred (56))* procedure2a.sql - B...inal (bby\Fred (55))*

```sql
CREATE PROCEDURE ADD_MOVIE_QUEUE
@mov_id_arg DECIMAL(12),
@cus_id_arg DECIMAL(12)
AS
BEGIN
UPDATE QUEUE_MOVIE
SET QUEUE_MOVIE.queue_priority = QUEUE_MOVIE.queue_priority + 1
WHERE customer_id = @cus_id_arg;

INSERT INTO QUEUE_MOVIE(movie_id, customer_id, queue_priority)
VALUES(@mov_id_arg, @cus_id_arg, 1);
END;
```

SQLQuery3.sql - BB...nal (bby\Fred (56))* procedure2a.sql - B...i

```sql
SELECT * FROM QUEUE_MOVIE;;

 EXECUTE ADD_MOVIE_QUEUE 25, 1001;

SELECT * FROM QUEUE_MOVIE;
```

100 %

Results Messages

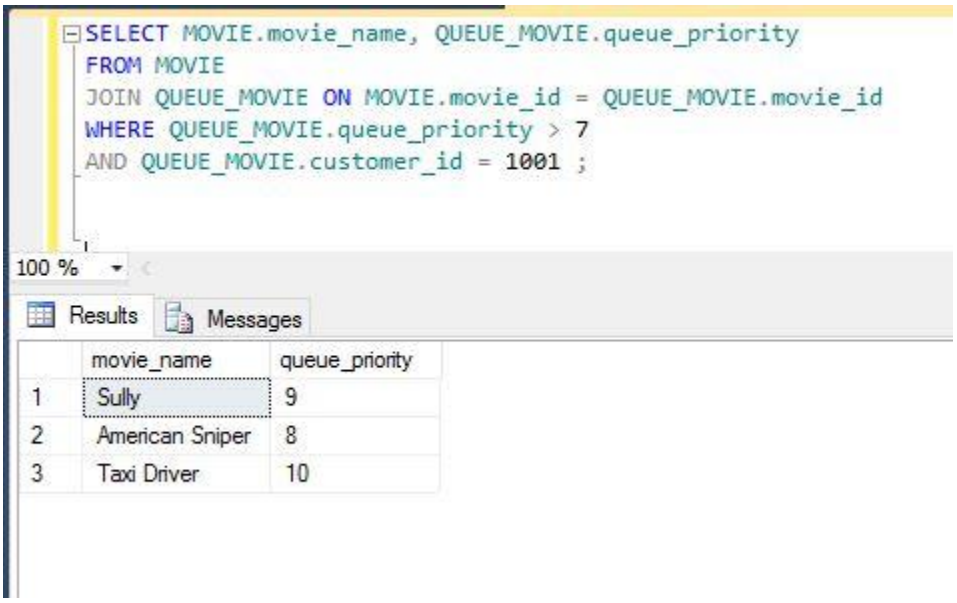
	queue_id	movie_id	customer_id	queue_priority
1	31	4	1001	9
2	32	9	1001	8
3	33	7	1001	7
4	34	6	1001	6
5	35	13	1001	5
6	36	21	1001	4
7	37	19	1001	3
8	38	8	1001	2
9	39	10	1001	1

	queue_id	movie_id	customer_id	queue_priority
1	31	4	1001	10
2	32	9	1001	9
3	33	7	1001	8
4	34	6	1001	7
5	35	13	1001	6
6	36	21	1001	5
7	37	19	1001	4
8	38	8	1001	3
9	39	10	1001	2
10	40	25	1001	1

b. A customer wants to see the names of the first three movies in their queue. Write a single query that retrieves this information for a customer of your choosing.

Code:

```
SELECT MOVIE.movie_name, QUEUE_MOVIE.queue_priority
FROM MOVIE
JOIN QUEUE_MOVIE ON MOVIE.movie_id = QUEUE_MOVIE.movie_id
WHERE QUEUE_MOVIE.queue_priority > 7
AND QUEUE_MOVIE.customer_id = 1001 ;
```



Use Case 3 – Enrolment:

a. A customer enrolls in the two-at-a-time plan and another customer enrolls in the three-at-a-time plan. Develop a parameterized stored procedure that accomplishes this, then invoke the stored procedure two times (to satisfy the use case) for customers of your choosing.

Code:

```
CREATE PROCEDURE ADD_MEMBER_ENROLLMENT
@cus_id_arg DECIMAL(12),
@mem_type_arg DECIMAL(12)
AS
```

```
INSERT INTO MEMBER (customer_id, membership_type_id, customer_date_start,
customer_date_end)
VALUES (@cus_id_arg, @mem_type_arg, GETDATE(), NULL);
```

SQLQuery2.sql - BB...ect (bby\Fred (57))* | insert_tables1.sql -...oject (bby\Fred (53)) | project_tables.sql -...ject (bby\Fred (52))

```
CREATE PROCEDURE ADD_MEMBER_ENROLLMENT
@cus_id_arg DECIMAL(12),
@mem_type_arg DECIMAL(12)
AS
INSERT INTO MEMBER (customer_id, membership_type_id, customer_date_start, customer_date_end)
VALUES (@cus_id_arg, @mem_type_arg, GETDATE(), NULL);
```

100 %

Messages

```
Command(s) completed successfully.
```

```
SELECT * FROM MEMBER;

EXECUTE ADD_MEMBER_ENROLLMENT 1001,1 ;
EXECUTE ADD_MEMBER_ENROLLMENT 1002,2 ;

SELECT * FROM MEMBER;
```

100 %

Results | Messages

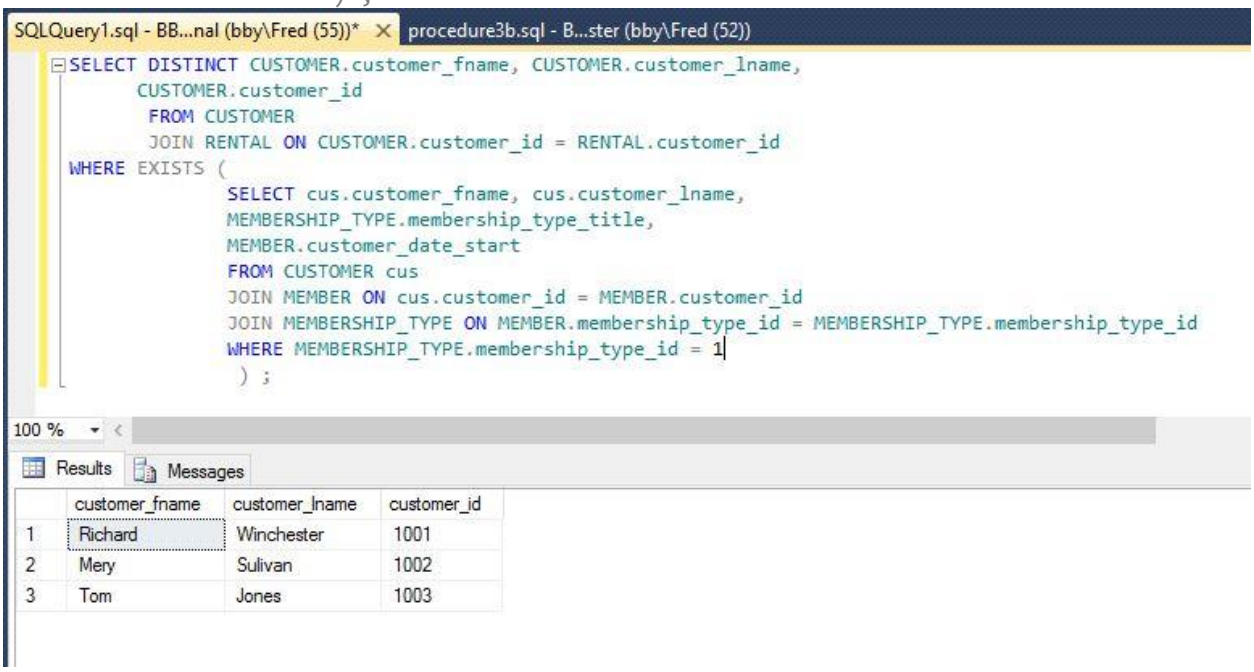
member_id	customer_id	membership_type_id	customer_date_start	customer_date_end

	member_id	customer_id	membership_type_id	customer_date_start	customer_date_end
1	4	1001	1	2017-02-19	NULL
2	5	1002	2	2017-02-19	NULL

b. Management requests the names of all currently active customers that are enrolled in the two-at-a-time plan. Write a single query that retrieves this information.

Code:

```
SELECT DISTINCT CUSTOMER.customer_fname, CUSTOMER.customer_lname,
      CUSTOMER.customer_id
          FROM CUSTOMER
          JOIN RENTAL ON CUSTOMER.customer_id = RENTAL.customer_id
WHERE EXISTS (
               SELECT cus.customer_fname, cus.customer_lname,
               MEMBERSHIP_TYPE.membership_type_title,
                  MEMBER.customer_date_start
               FROM CUSTOMER cus
               JOIN MEMBER ON cus.customer_id = MEMBER.customer_id
               JOIN MEMBERSHIP_TYPE ON MEMBER.membership_type_id =
MEMBERSHIP_TYPE.membership_type_id
               WHERE MEMBERSHIP_TYPE.membership_type_id = 1
                        ) ;
```



Use Case 4 – Inventory:

a. Netflix acquires three more copies of the movie “Star Trek Into Darkness”, and two more copies of “Dredd”, and a staff member records these inventory changes in the database. Develop a parameterized stored procedure and invoke it multiple times to satisfy the use case.

Code:

```
CREATE PROCEDURE ADD_MORE_COPIES
@movie_name_arg VARCHAR(128),
@dvd_copy_arg DECIMAL(6)
AS
BEGIN
UPDATE MOVIE
SET dvd_copy_number = dvd_copy_number + @dvd_copy_arg
WHERE movie_name = @movie_name_arg
END ;
```

SQLQuery2.sql - BB...nal (bby\Fred (56))* | project_tables.sql -...final (bby\Fred (

```
SELECT movie_name, dvd_copy_number FROM MOVIE ;

EXECUTE ADD_MORE_COPIES 'Star Trek Into Darkness', 3 ;
EXECUTE ADD_MORE_COPIES 'Dredd', 2 ;

SELECT movie_name, dvd_copy_number FROM MOVIE ;
```

100 %

Results | Messages

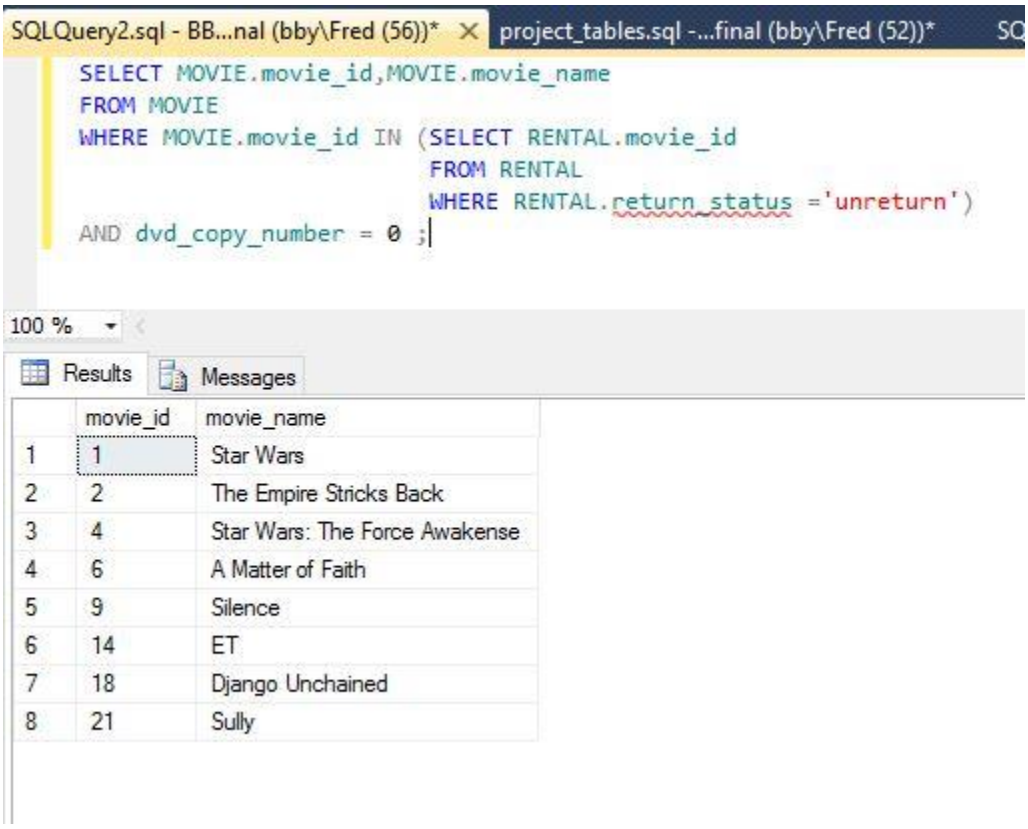
	movie_name	dvd_copy_number
21	Sully	0
22	Avatar	5
23	Titanic	3
24	Aliens	2
25	X-Men: Days of Future Past	6
26	The Angri Birds Movie	5
27	Star Trek Into Darkness	1
28	Dredd	1

	movie_name	dvd_copy_number
18	Django Unchained	0
19	American Sniper	4
20	Mystic River	4
21	Sully	0
22	Avatar	5
23	Titanic	3
24	Aliens	2
25	X-Men: Days of Future ...	6
26	The Angri Birds Movie	5
27	Star Trek Into Darkness	4
28	Dredd	3

b. Management requests the names of all movies that are currently sold out. A movie is sold out if all copies of the movie are currently rented and not yet returned. Write a single query that retrieves this information for management.

Code:

```
SELECT MOVIE.movie_id,MOVIE.movie_name
FROM MOVIE
WHERE MOVIE.movie_id IN (SELECT RENTAL.movie_id
                         FROM RENTAL
                         WHERE RENTAL.return_status ='unreturn')
AND dvd_copy_number = 0 ;
```



	movie_id	movie_name
1	1	Star Wars
2	2	The Empire Stricks Back
3	4	Star Wars: The Force Awakense
4	6	A Matter of Faith
5	9	Silence
6	14	ET
7	18	Django Unchained
8	21	Sully

Use Case 5 – Return Rental

a. A customer wants to return the movies ‘Star Wars’ and 'The Empire Stricks Back'. Develop a parameterized stored procedure that supports a customer returning a movie, then invoke the stored procedure two times (to satisfy the use case) for a customer of your choosing.

Code:

```
CREATE PROCEDURE RETURN_RENTAL_MOVIE
@cus_id_arg  DECIMAL(12),
@mov_name_arg VARCHAR(128)
AS
BEGIN

UPDATE RENTAL
SET rental_date_return = GETDATE(), return_status = 'return'
WHERE customer_id = @cus_id_arg
AND movie_id = (SELECT RENTAL.movie_id FROM RENTAL
                                        JOIN MOVIE ON RENTAL.movie_id = MOVIE.movie_id
                                                                        WHERE
MOVIE.movie_name = @mov_name_arg) ;

UPDATE MOVIE
SET dvd_copy_number = dvd_copy_number + 1
WHERE movie_name = @mov_name_arg ;
END ;

SELECT * FROM RENTAL;
EXECUTE RETURN_RENTAL_MOVIE 1001, 'Star Wars';
EXECUTE RETURN_RENTAL_MOVIE 1001, 'The Empire Stricks Back';
SELECT * FROM RENTAL;
```

SQLQuery3.sql - BB...nal (bby\Fred (57))* | project_tables.sql -...final (bby\Fred (53))* | procedure1a.sql

```sql
SELECT * FROM RENTAL;
EXECUTE RETURN_RENTAL_MOVIE 1001, 'Star Wars';
EXECUTE RETURN_RENTAL_MOVIE 1001, 'The Empire Stricks Back';
SELECT * FROM RENTAL;
```

100 %

Results | Messages

	rental_id	customer_id	movie_id	rental_date_delivery	rental_date_return	return_status
1	46	1001	25	2017-02-24	NULL	unreturn
2	47	1001	26	2017-02-24	NULL	unreturn
3	48	1001	1	2017-02-24	NULL	unreturn
4	49	1001	2	2017-02-24	NULL	unreturn
5	50	1001	3	2017-02-24	NULL	unreturn
6	51	1001	4	2017-02-24	NULL	unreturn
7	52	1001	5	2017-02-24	NULL	unreturn
8	53	1001	6	2017-02-24	NULL	unreturn
9	54	1003	5	2017-02-24	NULL	unreturn
10	55	1003	14	2017-02-24	NULL	unreturn
11	56	1002	9	2017-02-24	NULL	unreturn

	rental_id	customer_id	movie_id	rental_date_delivery	rental_date_return	return_status
1	46	1001	25	2017-02-24	NULL	unreturn
2	47	1001	26	2017-02-24	NULL	unreturn
3	48	1001	1	2017-02-24	2017-02-25	return
4	49	1001	2	2017-02-24	2017-02-25	return
5	50	1001	3	2017-02-24	NULL	unreturn
6	51	1001	4	2017-02-24	NULL	unreturn
7	52	1001	5	2017-02-24	NULL	unreturn
8	53	1001	6	2017-02-24	NULL	unreturn
9	54	1003	5	2017-02-24	NULL	unreturn
10	55	1003	14	2017-02-24	NULL	unreturn

b. Management requests the names of all movies that are rented by multi times. Write a single query that retrieves this information for management.

Code:

```sql
SELECT RENTAL.movie_id, MOVIE.movie_name, COUNT(DISTINCT CUSTOMER.customer_id) AS
NUM_OF_RENTED
FROM RENTAL
JOIN CUSTOMER ON CUSTOMER.customer_id = RENTAL.customer_id
JOIN MOVIE ON  MOVIE.movie_id = RENTAL.movie_id
GROUP BY RENTAL.movie_id, MOVIE.movie_name
HAVING COUNT(DISTINCT CUSTOMER.customer_id) > 1 ;
```

procedure5b.sql - B...inal (bby\Fred (53))* | procedure5a.sql - B...inal (bby\Fred (52))

```sql
SELECT RENTAL.movie_id, MOVIE.movie_name, COUNT(DISTINCT CUSTOMER.customer_id) AS NUM_OF_RENTED
FROM RENTAL
JOIN CUSTOMER ON CUSTOMER.customer_id = RENTAL.customer_id
JOIN MOVIE ON  MOVIE.movie_id = RENTAL.movie_id
GROUP BY RENTAL.movie_id, MOVIE.movie_name
HAVING COUNT(DISTINCT CUSTOMER.customer_id) > 1 ;
```

100 %

Results | Messages

	movie_id	movie_name	NUM_OF_RENTED
1	5	Time Changer	2

Indexes:

I created two indexes on MOVIE.movie_name and DIRECTOR.director_lname as non-unique indexes, because these columns are indicated by where clause and making index on these columns can improve speed up performance of query.

Object Explorer

Connect

- System Tables
- FileTables
- dbo.ADDRESS
- dbo.CUSTOMER
- dbo.DIRECTOR
  - Columns
  - Keys
  - Constraints
  - Triggers
  - Indexes
    - index_director_lname (Non-Unique, Non-Clustered)
    - PK__DIRECTOR__F5205E49258FE55E (Clustered)
  - Statistics
- dbo.GENRE
- dbo.MEMBER
- dbo.MEMBERSHIP_TYPE
- dbo.MOVIE
  - Columns
  - Keys
  - Constraints
  - Triggers
  - Indexes
    - index_movie_name (Non-Unique, Non-Clustered)
    - PK__MOVIE__83CDF749EA3B8AC0 (Clustered)

index.sql - BBY\SQL...inal (bby\Fred (52))*

```sql
CREATE INDEX index_movie_name
ON MOVIE (movie_name) ;

CREATE INDEX index_director_lname
ON DIRECTOR (director_lname) ;
```

100 %

Messages

```
Command(s) completed successfully.
```