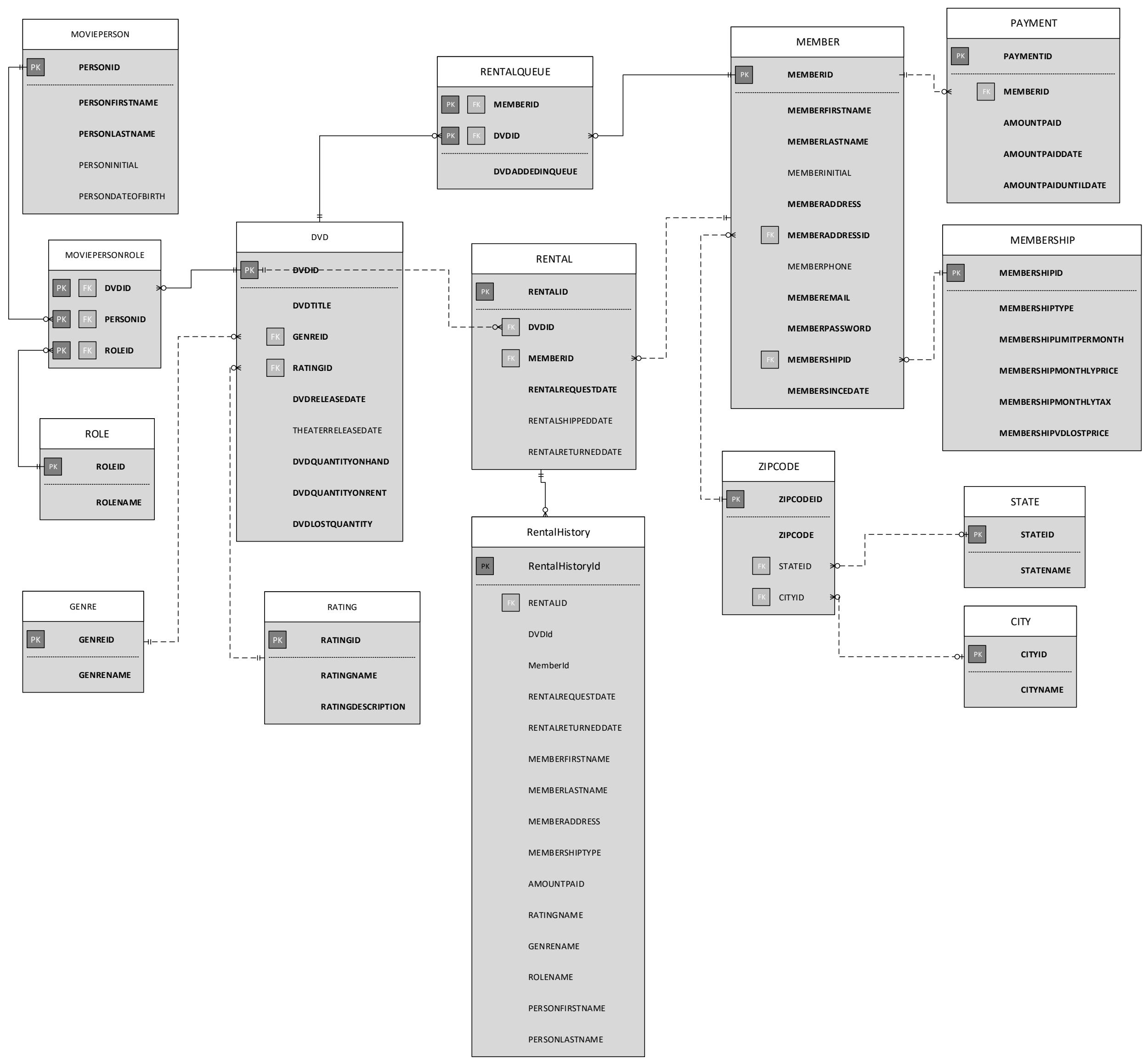
Sample SQL codes (Procedures, Triggers and Functions) in Advanced Database Management Course

Fariborz Norouzi

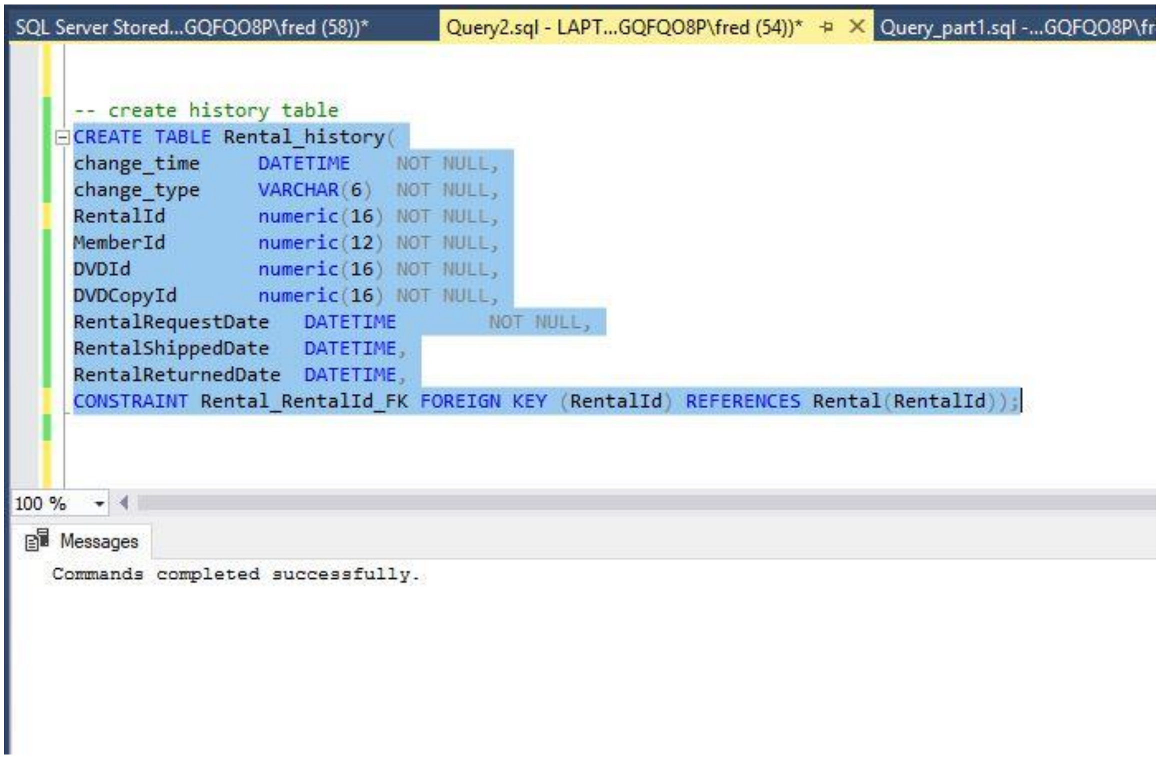
Summer 2018



1. Evaluate and implement (create) a **History** table for the Netflix database.

A Rental history table allows us to use to track changes in Rental table. It tends to be very useful to track what action caused the history entry, be it an INSERT, UPDATE, or DELETE.

```
-- create history table
CREATE TABLE Rental_history(
change_time             DATETIME    NOT NULL,
change_type             VARCHAR(6)    NOT NULL,
RentalId                numeric(16)   NOT NULL,
MemberId                numeric(12)   NOT NULL,
DVDId                   numeric(16)   NOT NULL,
DVDCopyId         numeric(16) NOT NULL,
RentalRequestDate     DATETIME            NOT NULL,
RentalShippedDate     DATETIME,
RentalReturnedDate    DATETIME,
    CONSTRAINT Rental_RentalId_FK FOREIGN KEY (RentalId) REFERENCES Rental(RentalId));
```

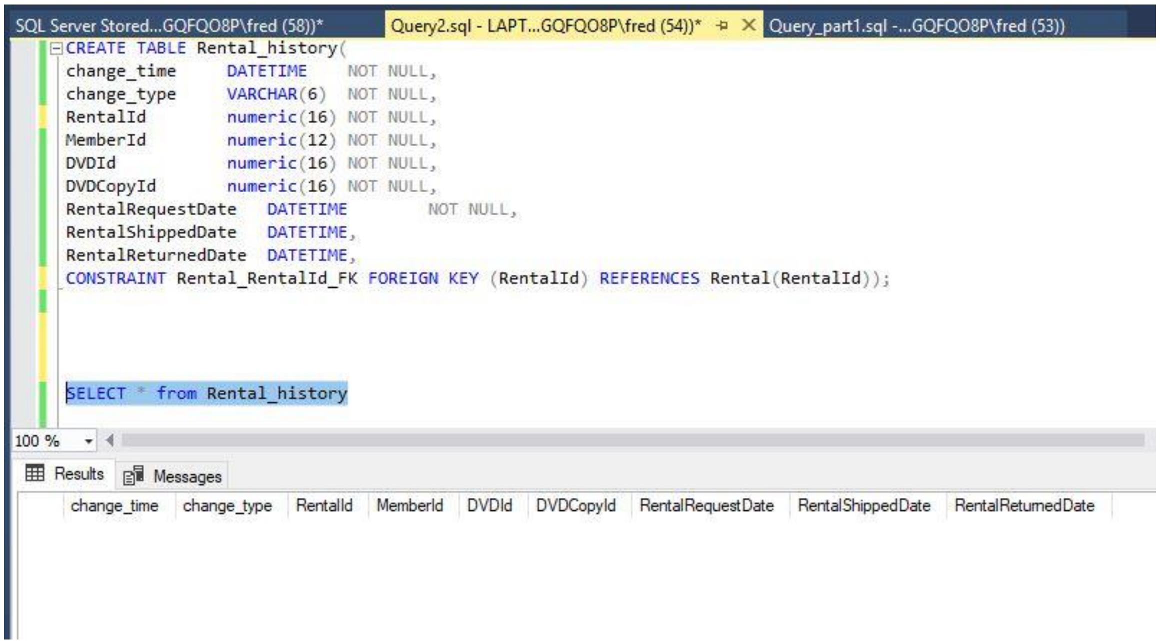

SQL Server Stored...GQFQO8P\fred (58))* | Query2.sql - LAPT...GQFQO8P\fred (54))* | Query_part1.sql -...GQFQO8P\fr

```
-- create history table
CREATE TABLE Rental_history(
change_time       DATETIME     NOT NULL,
change_type       VARCHAR(6)   NOT NULL,
RentalId          numeric(16)  NOT NULL,
MemberId          numeric(12)  NOT NULL,
DVDId             numeric(16)  NOT NULL,
DVDCopyId         numeric(16)  NOT NULL,
RentalRequestDate    DATETIME        NOT NULL,
RentalShippedDate    DATETIME,
RentalReturnedDate   DATETIME,
CONSTRAINT Rental_RentalId_FK FOREIGN KEY (RentalId) REFERENCES Rental(RentalId));
```

100 %

Messages

```
Commands completed successfully.
```


SQL Server Stored...GQFQO8P\fred (58))* | Query2.sql - LAPT...GQFQO8P\fred (54))* | Query_part1.sql -...GQFQO8P\fred (53))

```
CREATE TABLE Rental_history(
change_time       DATETIME     NOT NULL,
change_type       VARCHAR(6)   NOT NULL,
RentalId          numeric(16)  NOT NULL,
MemberId          numeric(12)  NOT NULL,
DVDId             numeric(16)  NOT NULL,
DVDCopyId         numeric(16)  NOT NULL,
RentalRequestDate    DATETIME        NOT NULL,
RentalShippedDate    DATETIME,
RentalReturnedDate   DATETIME,
CONSTRAINT Rental_RentalId_FK FOREIGN KEY (RentalId) REFERENCES Rental(RentalId));



SELECT * from Rental_history
```

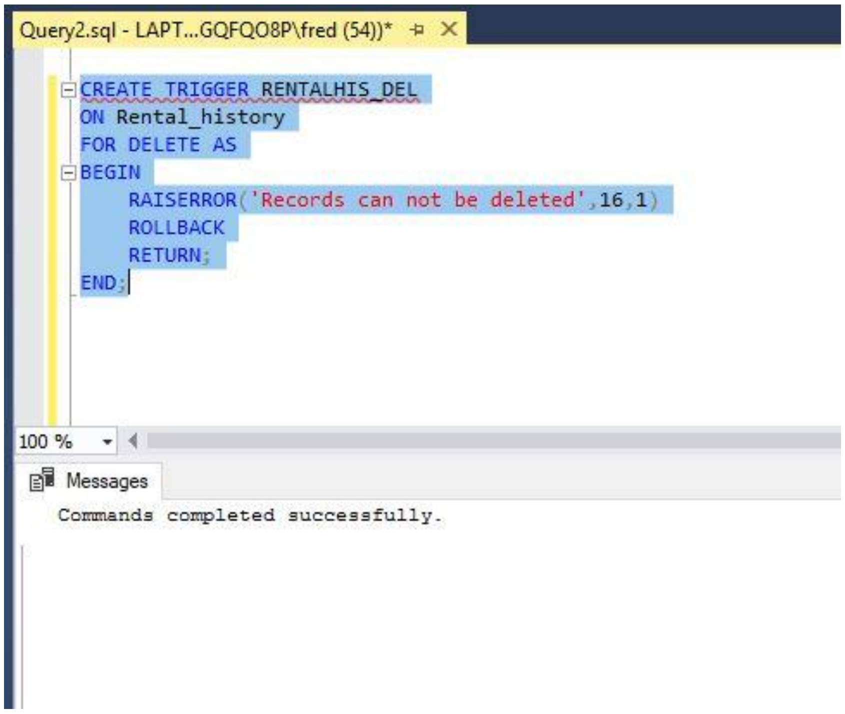
100 %

Results | Messages

change_time	change_type	RentalId	MemberId	DVDId	DVDCopyId	RentalRequestDate	RentalShippedDate	RentalReturnedDate

2. Implement a **trigger** for this new **rental history** table that prevents deletions from the table.

```
CREATE TRIGGER RENTALHIS_DEL
ON Rental_history
FOR DELETE AS
BEGIN
    RAISERROR('Records can not be deleted',16,1)
       ROLLBACK
       RETURN;
END;

DELETE FROM Rental_history ;
```



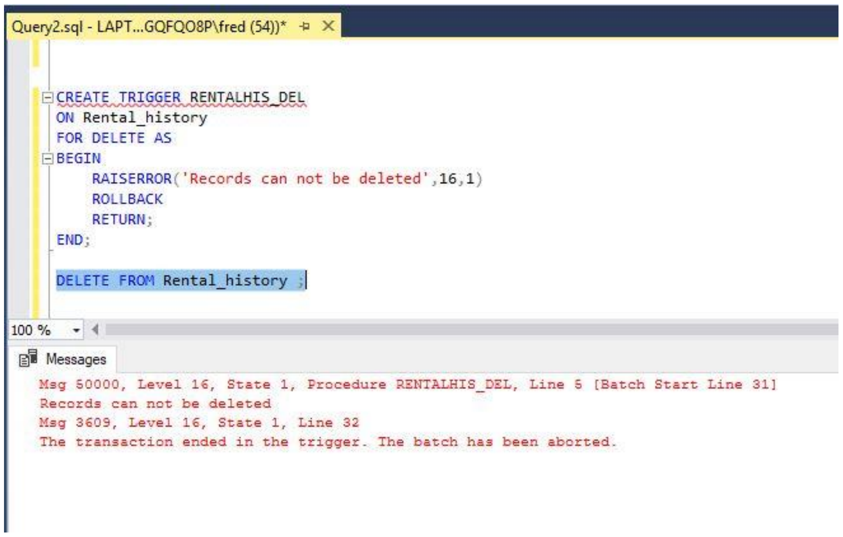
Query2.sql - LAPT...GQFQO8P\fred (54))*

```
CREATE TRIGGER RENTALHIS_DEL
ON Rental_history
FOR DELETE AS
BEGIN
    RAISERROR('Records can not be deleted',16,1)
    ROLLBACK
    RETURN;
END;
```

100 %

Messages

```
Commands completed successfully.
```



Query2.sql - LAPT...GQFQO8P\fred (54))*

```
CREATE TRIGGER RENTALHIS_DEL
ON Rental_history
FOR DELETE AS
BEGIN
    RAISERROR('Records can not be deleted',16,1)
    ROLLBACK
    RETURN;
END;

DELETE FROM Rental_history ;
```

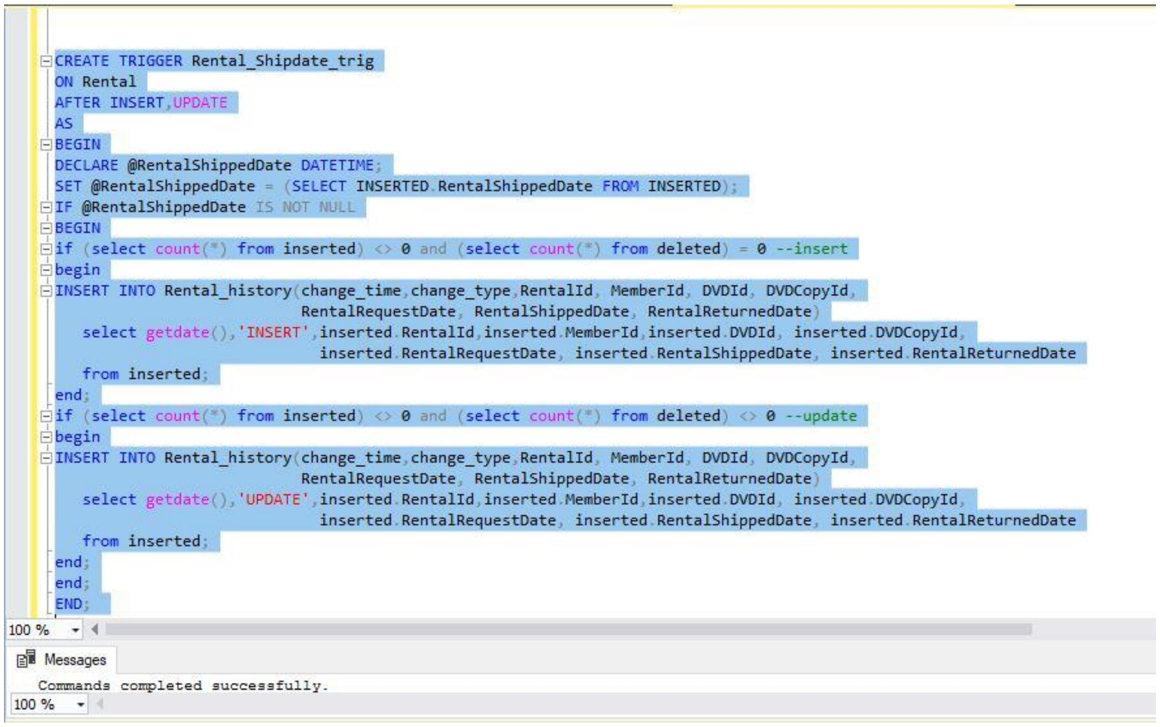
100 %

Messages

```
Msg 50000, Level 16, State 1, Procedure RENTALHIS_DEL, Line 5 [Batch Start Line 31]
Records can not be deleted
Msg 3609, Level 16, State 1, Line 32
The transaction ended in the trigger. The batch has been aborted.
```

3. Implement a **trigger** that automatically updates the **rental history** when a DVD is shipped to a customer. Depending on your design of the rental history, this update may be an UPDATE and/or an INSERT.

```sql
CREATE TRIGGER Rental_Shipdate_trig
ON Rental
AFTER INSERT,UPDATE
AS
BEGIN
DECLARE @RentalShippedDate DATETIME;
SET @RentalShippedDate = (SELECT INSERTED.RentalShippedDate FROM INSERTED);
IF @RentalShippedDate IS NOT NULL
BEGIN
if (select count(*) from inserted) <> 0 and (select count(*) from deleted) = 0 --insert
begin

INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'INSERT',inserted.RentalId,inserted.MemberId,inserted.DVDId,
inserted.DVDCopyId, inserted.RentalRequestDate, inserted.RentalShippedDate,
inserted.RentalReturnedDate  from inserted;
end;

if (select count(*) from inserted) <> 0 and (select count(*) from deleted) <> 0 --update
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'UPDATE',inserted.RentalId,inserted.MemberId,inserted.DVDId,
inserted.DVDCopyId, inserted.RentalRequestDate, inserted.RentalShippedDate,
inserted.RentalReturnedDate
   from inserted;
end;
end;
END;
```

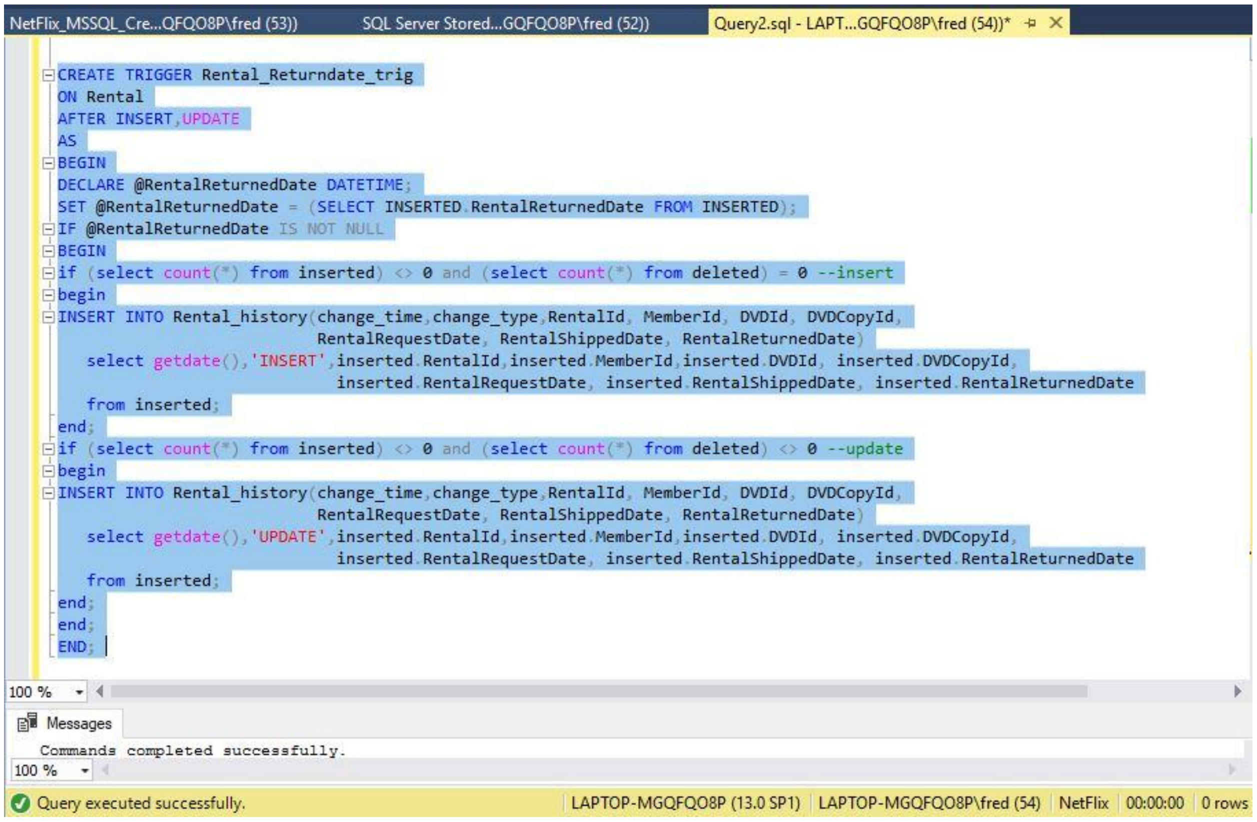

```
CREATE TRIGGER Rental_Shipdate_trig
ON Rental
AFTER INSERT,UPDATE
AS
BEGIN
DECLARE @RentalShippedDate DATETIME;
SET @RentalShippedDate = (SELECT INSERTED.RentalShippedDate FROM INSERTED);
IF @RentalShippedDate IS NOT NULL
BEGIN
if (select count(*) from inserted) <> 0 and (select count(*) from deleted) = 0 --insert
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
                            RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'INSERT',inserted.RentalId,inserted.MemberId,inserted.DVDId, inserted.DVDCopyId,
                            inserted.RentalRequestDate, inserted.RentalShippedDate, inserted.RentalReturnedDate
   from inserted;
end;
if (select count(*) from inserted) <> 0 and (select count(*) from deleted) <> 0 --update
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
                            RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'UPDATE',inserted.RentalId,inserted.MemberId,inserted.DVDId, inserted.DVDCopyId,
                            inserted.RentalRequestDate, inserted.RentalShippedDate, inserted.RentalReturnedDate
   from inserted;
end;
end;
END;
```
100 %

Messages

```
Commands completed successfully.
```
100 %

4. Implement a second **trigger** that automatically updates the rental history when a DVD is received from a customer.

```
CREATE TRIGGER Rental_Returndate_trig
ON Rental
AFTER INSERT,UPDATE
AS
BEGIN
DECLARE @RentalReturnedDate DATETIME;
SET @RentalReturnedDate = (SELECT INSERTED.RentalReturnedDate FROM INSERTED);
IF @RentalReturnedDate IS NOT NULL
BEGIN
if (select count(*) from inserted) <> 0 and (select count(*) from deleted) = 0 --insert
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
                           RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'INSERT',inserted.RentalId,inserted.MemberId,inserted.DVDId,
inserted.DVDCopyId, inserted.RentalRequestDate, inserted.RentalShippedDate,
inserted.RentalReturnedDate from inserted;
end;

if (select count(*) from inserted) <> 0 and (select count(*) from deleted) <> 0 --update
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
                           RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'UPDATE',inserted.RentalId,inserted.MemberId,inserted.DVDId,
inserted.DVDCopyId, inserted.RentalRequestDate, inserted.RentalShippedDate,
inserted.RentalReturnedDate  from inserted;
end;
end;
END;
```



NetFlix_MSSQL_Cre...QFQO8P\fred (53))   SQL Server Stored...GQFQO8P\fred (52))   Query2.sql - LAPT...GQFQO8P\fred (54))*

```
CREATE TRIGGER Rental_Returndate_trig
ON Rental
AFTER INSERT,UPDATE
AS
BEGIN
DECLARE @RentalReturnedDate DATETIME;
SET @RentalReturnedDate = (SELECT INSERTED.RentalReturnedDate FROM INSERTED);
IF @RentalReturnedDate IS NOT NULL
BEGIN
if (select count(*) from inserted) <> 0 and (select count(*) from deleted) = 0 --insert
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
                           RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'INSERT',inserted.RentalId,inserted.MemberId,inserted.DVDId, inserted.DVDCopyId,
                             inserted.RentalRequestDate, inserted.RentalShippedDate, inserted.RentalReturnedDate
   from inserted;
end;
if (select count(*) from inserted) <> 0 and (select count(*) from deleted) <> 0 --update
begin
INSERT INTO Rental_history(change_time,change_type,RentalId, MemberId, DVDId, DVDCopyId,
                           RentalRequestDate, RentalShippedDate, RentalReturnedDate)
   select getdate(),'UPDATE',inserted.RentalId,inserted.MemberId,inserted.DVDId, inserted.DVDCopyId,
                             inserted.RentalRequestDate, inserted.RentalShippedDate, inserted.RentalReturnedDate
   from inserted;
end;
end;
END;
```

100 %

Messages

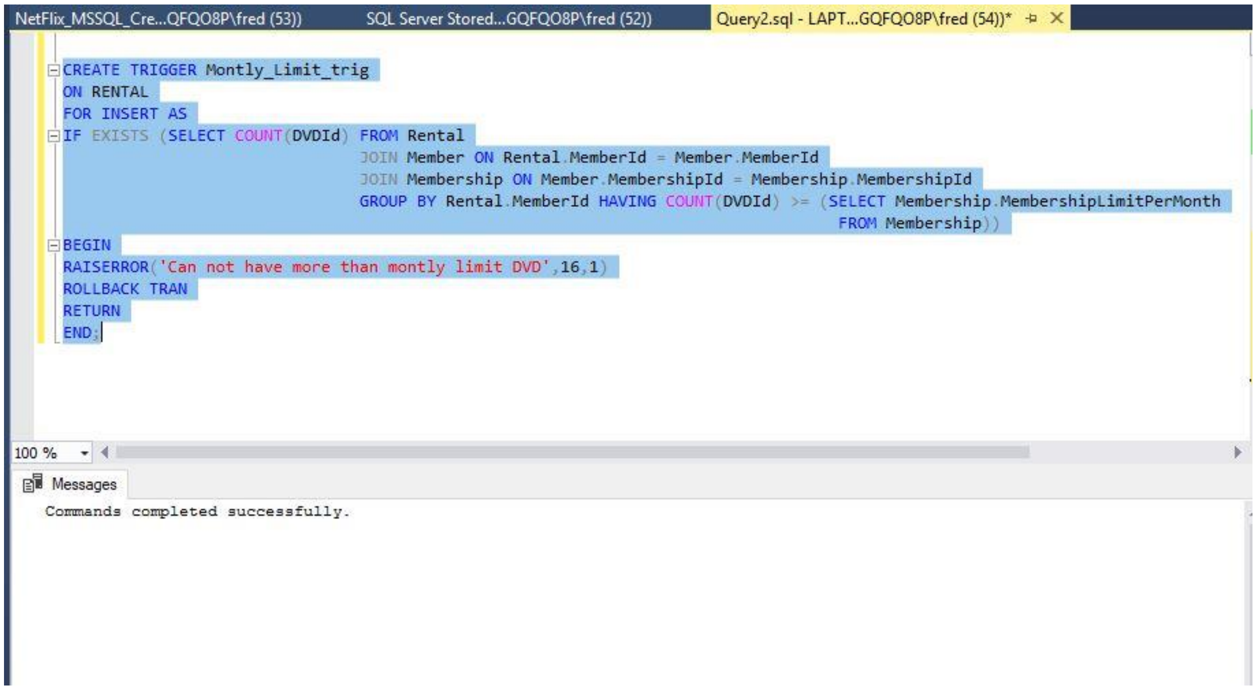
```
Commands completed successfully.
```

100 %

Query executed successfully.   LAPTOP-MGQFQO8P (13.0 SP1) | LAPTOP-MGQFQO8P\fred (54) | NetFlix | 00:00:00 | 0 rows

5. Implement a **trigger** that prevents a customer from being shipped a DVD if they have reached their monthly limit for DVD rentals as per their membership contract.

```
CREATE TRIGGER Montly_Limit_trig
ON RENTAL
FOR INSERT AS
IF EXISTS (SELECT COUNT(DVDId) FROM Rental
                                JOIN Member ON Rental.MemberId = Member.MemberId
                               JOIN Membership ON Member.MembershipId =
Membership.MembershipId
                               GROUP BY Rental.MemberId HAVING COUNT(DVDId) >= (SELECT
Membership.MembershipLimitPerMonth FROM Membership))
BEGIN
RAISERROR('Can not have more than montly limit DVD',16,1)
ROLLBACK TRAN
RETURN
END;
```



NetFlix_MSSQL_Cre...QFQO8P\fred (53))   SQL Server Stored...GQFQO8P\fred (52))   Query2.sql - LAPT...GQFQO8P\fred (54))*

```
CREATE TRIGGER Montly_Limit_trig
ON RENTAL
FOR INSERT AS
IF EXISTS (SELECT COUNT(DVDId) FROM Rental
                              JOIN Member ON Rental.MemberId = Member.MemberId
                              JOIN Membership ON Member.MembershipId = Membership.MembershipId
                              GROUP BY Rental.MemberId HAVING COUNT(DVDId) >= (SELECT Membership.MembershipLimitPerMonth
                                                                                  FROM Membership))
BEGIN
RAISERROR('Can not have more than montly limit DVD',16,1)
ROLLBACK TRAN
RETURN
END;
```

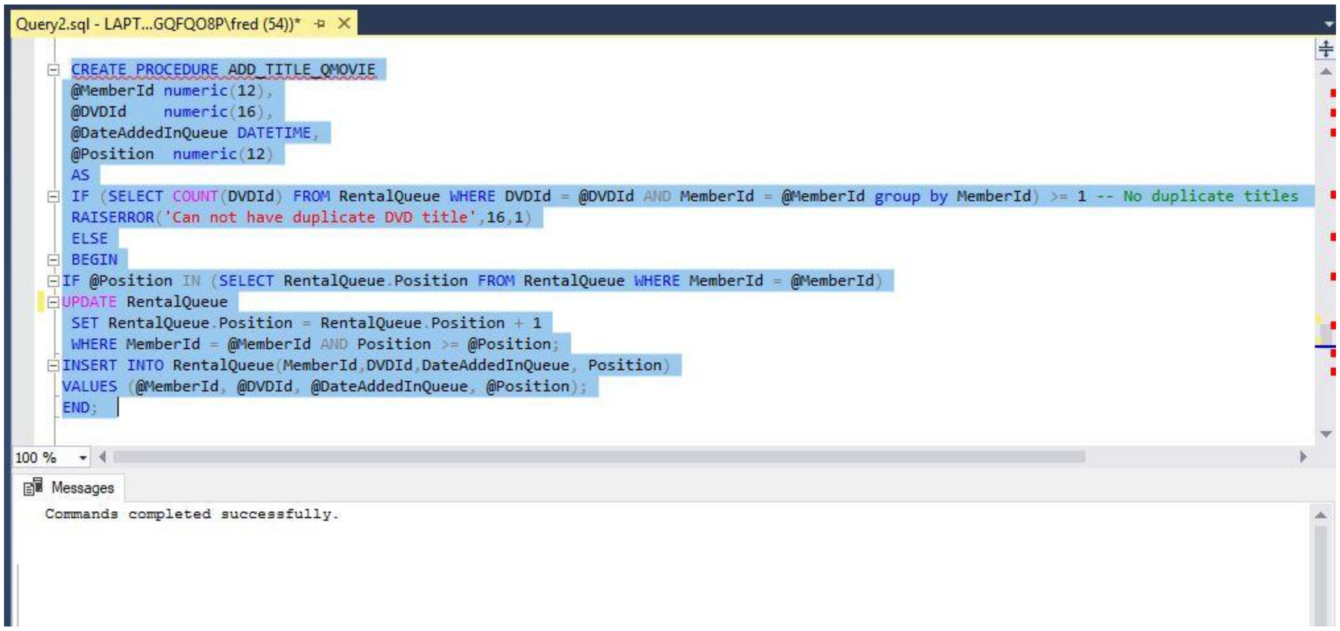
100 %

Messages

```
Commands completed successfully.
```

6. Implement a **stored procedure** that adds a title to the customer's movie list (the **Rental Queue** table). This procedure should take as IN parameters the customer ID and movie title ID as well as the location of where the movie is in the queue. The procedure should also make sure that no duplicate titles can be added.

```
CREATE PROCEDURE ADD_TITLE_QMOVIE
 @MemberId numeric(12),
 @DVDId     numeric(16),
 @DateAddedInQueue DATETIME,
 @Position  numeric(12)
 AS
 IF (SELECT COUNT(DVDId) FROM RentalQueue WHERE DVDId = @DVDId AND MemberId = @MemberId
group by MemberId) >= 1 -- No duplicate titles
 RAISERROR('Can not have duplicate DVD title',16,1)
 ELSE
 BEGIN
IF @Position IN (SELECT RentalQueue.Position FROM RentalQueue WHERE MemberId = @MemberId)
UPDATE RentalQueue
 SET RentalQueue.Position = RentalQueue.Position + 1
 WHERE MemberId = @MemberId AND Position >= @Position;
INSERT INTO RentalQueue(MemberId,DVDId,DateAddedInQueue, Position)
VALUES (@MemberId, @DVDId, @DateAddedInQueue, @Position);
   END;
```

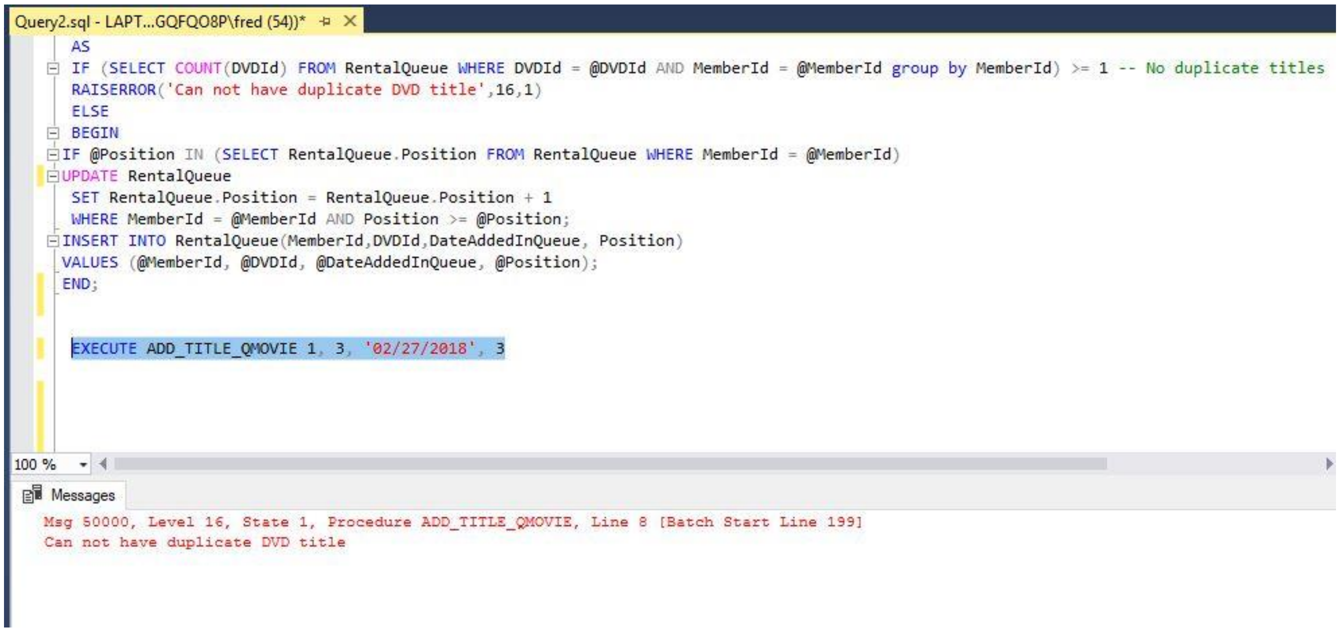

Query2.sql - LAPT...GQFQO8P\fred (54))*

```
CREATE PROCEDURE ADD_TITLE_QMOVIE
@MemberId numeric(12),
@DVDId     numeric(16),
@DateAddedInQueue DATETIME,
@Position  numeric(12)
AS
IF (SELECT COUNT(DVDId) FROM RentalQueue WHERE DVDId = @DVDId AND MemberId = @MemberId group by MemberId) >= 1 -- No duplicate titles
RAISERROR('Can not have duplicate DVD title',16,1)
ELSE
BEGIN
IF @Position IN (SELECT RentalQueue.Position FROM RentalQueue WHERE MemberId = @MemberId)
UPDATE RentalQueue
SET RentalQueue.Position = RentalQueue.Position + 1
WHERE MemberId = @MemberId AND Position >= @Position;
INSERT INTO RentalQueue(MemberId,DVDId,DateAddedInQueue, Position)
VALUES (@MemberId, @DVDId, @DateAddedInQueue, @Position);
END;
```

100 %

Messages

```
Commands completed successfully.
```

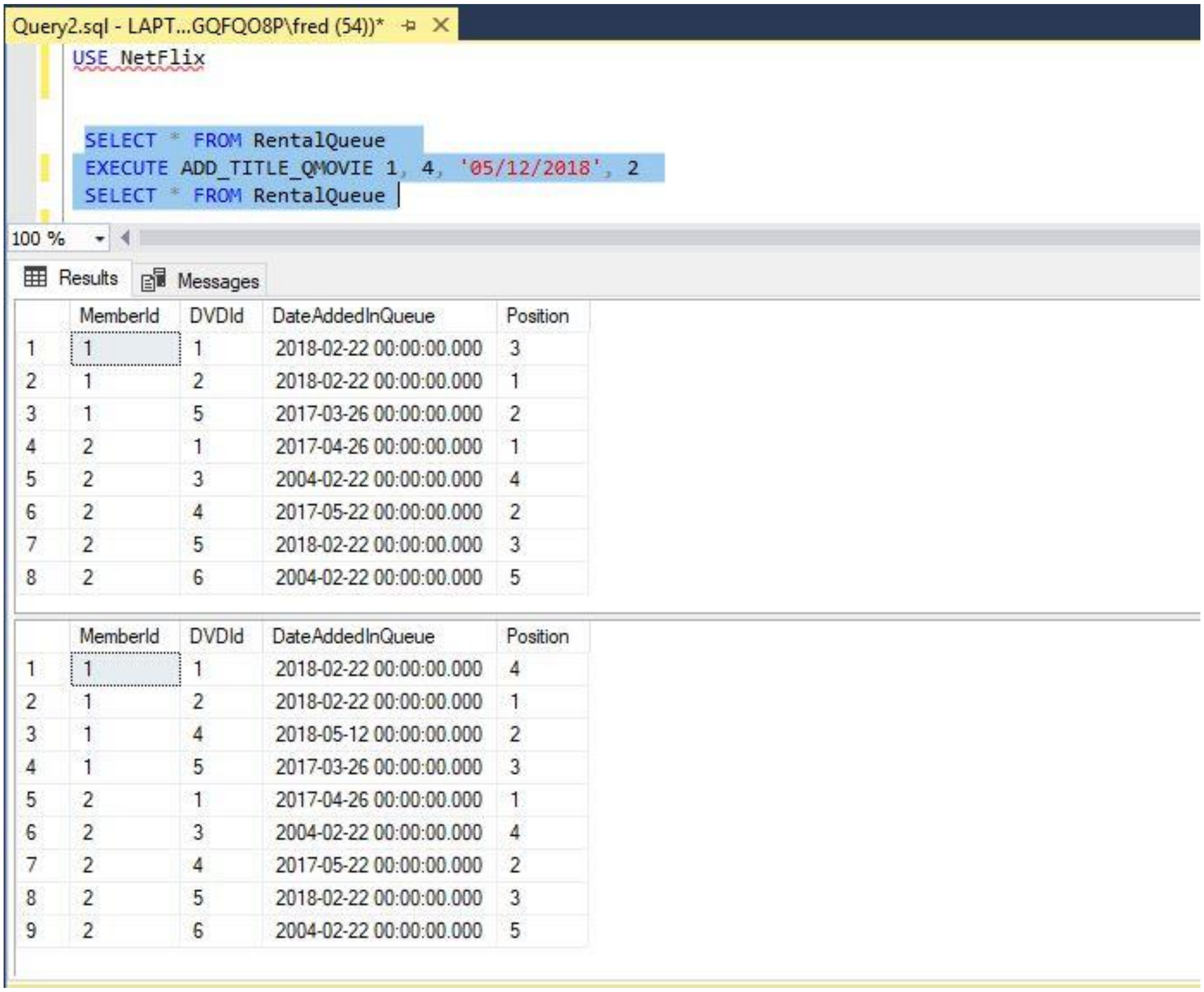

Query2.sql - LAPT...GQFQO8P\fred (54))*

```
AS
IF (SELECT COUNT(DVDId) FROM RentalQueue WHERE DVDId = @DVDId AND MemberId = @MemberId group by MemberId) >= 1 -- No duplicate titles
RAISERROR('Can not have duplicate DVD title',16,1)
ELSE
BEGIN
IF @Position IN (SELECT RentalQueue.Position FROM RentalQueue WHERE MemberId = @MemberId)
UPDATE RentalQueue
SET RentalQueue.Position = RentalQueue.Position + 1
WHERE MemberId = @MemberId AND Position >= @Position;
INSERT INTO RentalQueue(MemberId,DVDId,DateAddedInQueue, Position)
VALUES (@MemberId, @DVDId, @DateAddedInQueue, @Position);
END;


EXECUTE ADD_TITLE_QMOVIE 1, 3, '02/27/2018', 3
```

100 %

Messages

```
Msg 50000, Level 16, State 1, Procedure ADD_TITLE_QMOVIE, Line 8 [Batch Start Line 199]
Can not have duplicate DVD title
```



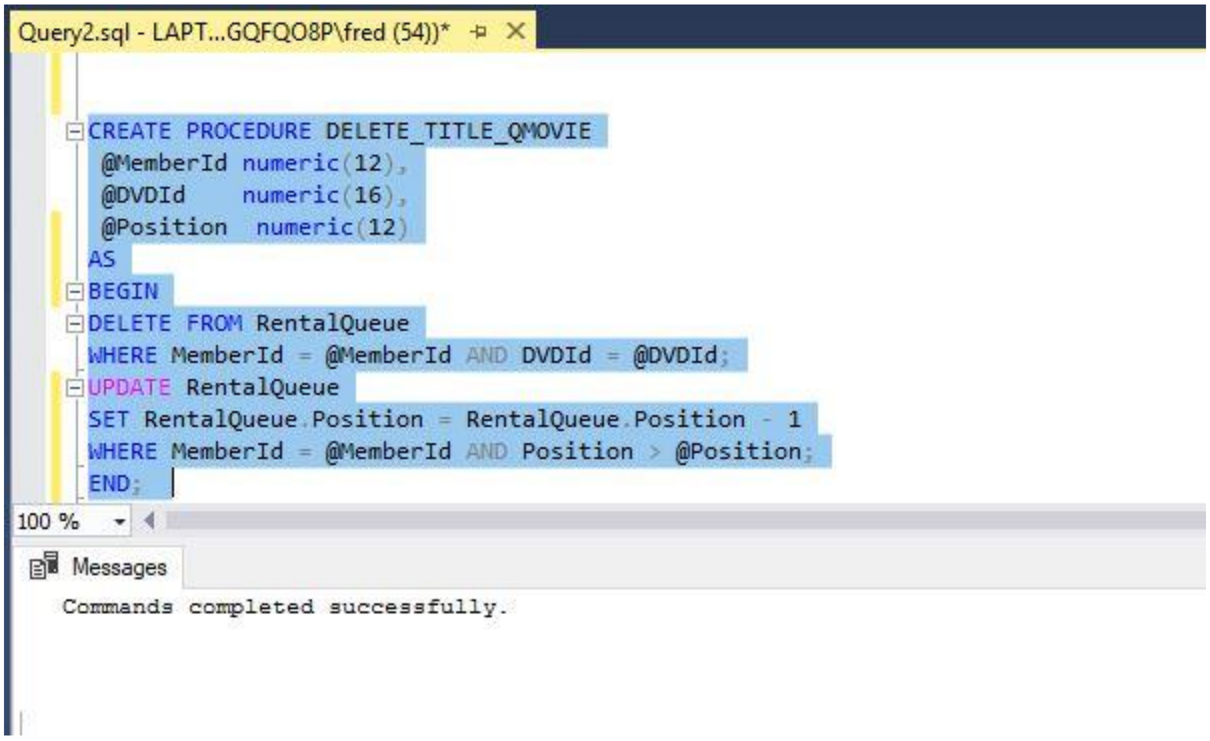
```
USE NetFlix


 SELECT * FROM RentalQueue
 EXECUTE ADD_TITLE_QMOVIE 1, 4, '05/12/2018', 2
 SELECT * FROM RentalQueue
```

	MemberId	DVDId	DateAddedInQueue	Position
1	1	1	2018-02-22 00:00:00.000	3
2	1	2	2018-02-22 00:00:00.000	1
3	1	5	2017-03-26 00:00:00.000	2
4	2	1	2017-04-26 00:00:00.000	1
5	2	3	2004-02-22 00:00:00.000	4
6	2	4	2017-05-22 00:00:00.000	2
7	2	5	2018-02-22 00:00:00.000	3
8	2	6	2004-02-22 00:00:00.000	5

	MemberId	DVDId	DateAddedInQueue	Position
1	1	1	2018-02-22 00:00:00.000	4
2	1	2	2018-02-22 00:00:00.000	1
3	1	4	2018-05-12 00:00:00.000	2
4	1	5	2017-03-26 00:00:00.000	3
5	2	1	2017-04-26 00:00:00.000	1
6	2	3	2004-02-22 00:00:00.000	4
7	2	4	2017-05-22 00:00:00.000	2
8	2	5	2018-02-22 00:00:00.000	3
9	2	6	2004-02-22 00:00:00.000	5

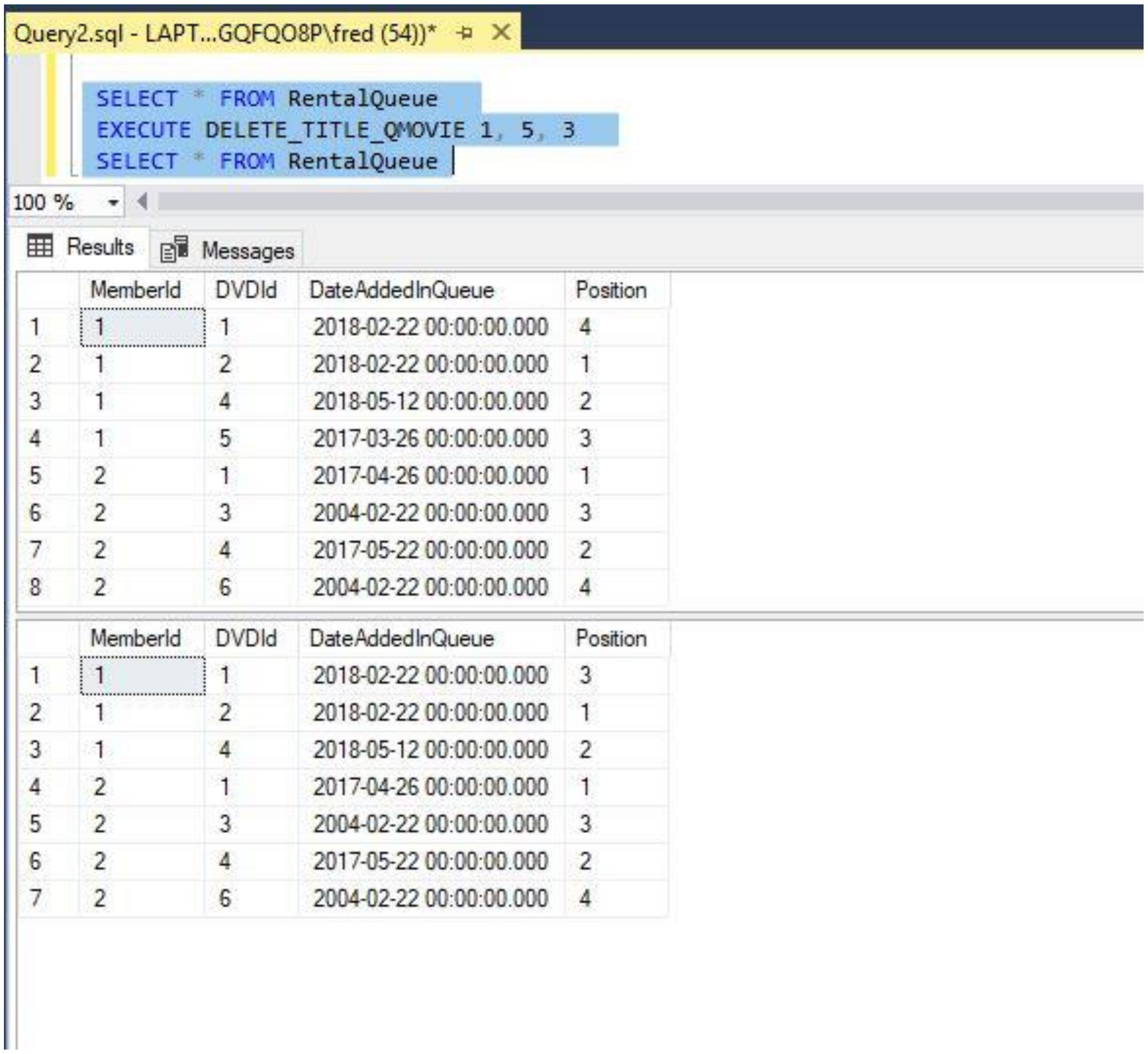
7. Write a **stored procedure** that deletes a title from a customer's movie list (the **Rental Queue** table). This procedure should take as IN parameters the customer ID and movie title ID.

```
CREATE PROCEDURE DELETE_TITLE_QMOVIE
 @MemberId numeric(12),
 @DVDId     numeric(16),
 @Position  numeric(12)
AS
BEGIN
DELETE FROM RentalQueue
WHERE MemberId = @MemberId AND DVDId = @DVDId;
UPDATE RentalQueue
SET RentalQueue.Position = RentalQueue.Position - 1
WHERE MemberId = @MemberId AND Position > @Position;
   END;
```


Query2.sql - LAPT...GQFQO8P\fred (54))*

```
CREATE PROCEDURE DELETE_TITLE_QMOVIE
 @MemberId numeric(12),
 @DVDId     numeric(16),
 @Position  numeric(12)
AS
BEGIN
DELETE FROM RentalQueue
WHERE MemberId = @MemberId AND DVDId = @DVDId;
UPDATE RentalQueue
SET RentalQueue.Position = RentalQueue.Position - 1
WHERE MemberId = @MemberId AND Position > @Position;
END;
```

```
Commands completed successfully.
```



Query2.sql - LAPT...GQFQO8P\fred (54))*

```
SELECT * FROM RentalQueue
EXECUTE DELETE_TITLE_QMOVIE 1, 5, 3
SELECT * FROM RentalQueue
```

100 %

Results | Messages

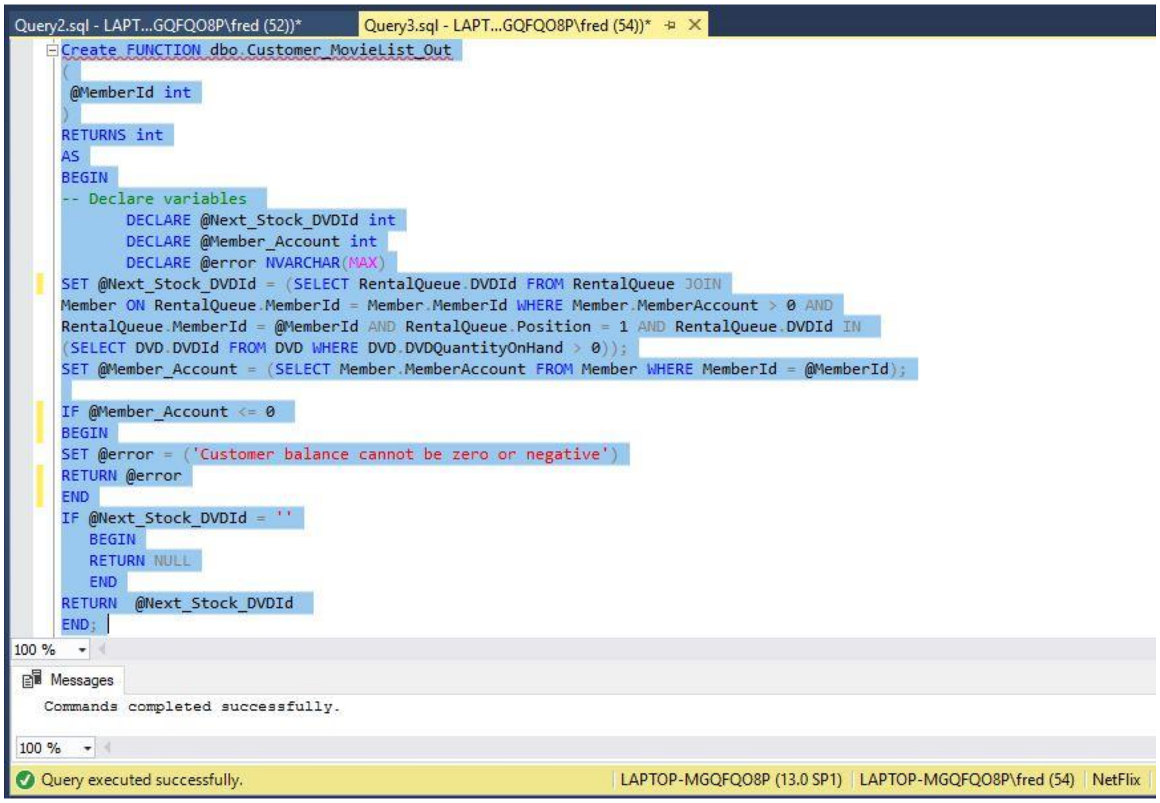
	MemberId	DVDId	DateAddedInQueue	Position
1	1	1	2018-02-22 00:00:00.000	4
2	1	2	2018-02-22 00:00:00.000	1
3	1	4	2018-05-12 00:00:00.000	2
4	1	5	2017-03-26 00:00:00.000	3
5	2	1	2017-04-26 00:00:00.000	1
6	2	3	2004-02-22 00:00:00.000	3
7	2	4	2017-05-22 00:00:00.000	2
8	2	6	2004-02-22 00:00:00.000	4

	MemberId	DVDId	DateAddedInQueue	Position
1	1	1	2018-02-22 00:00:00.000	3
2	1	2	2018-02-22 00:00:00.000	1
3	1	4	2018-05-12 00:00:00.000	2
4	2	1	2017-04-26 00:00:00.000	1
5	2	3	2004-02-22 00:00:00.000	3
6	2	4	2017-05-22 00:00:00.000	2
7	2	6	2004-02-22 00:00:00.000	4

1. Write a **function** that returns the DVD ID of the next in stock DVD in the customer's movie list (rental queue), and deletes that entry from the movie list (rental queue). The function should take as an IN parameter the customer ID and should return the DVD ID as the function value. The function should return the first title (the DVD ID) in the customer's movie list that is in stock. If the movie list is empty or none of the titles are in stock, then the function should return NULL. Additionally the function should perform error handling to check that the customer's account is valid for this operation.
    a. **Note:** If you are having a hard time implementing this as a function, you may implement this as a stored procedure.

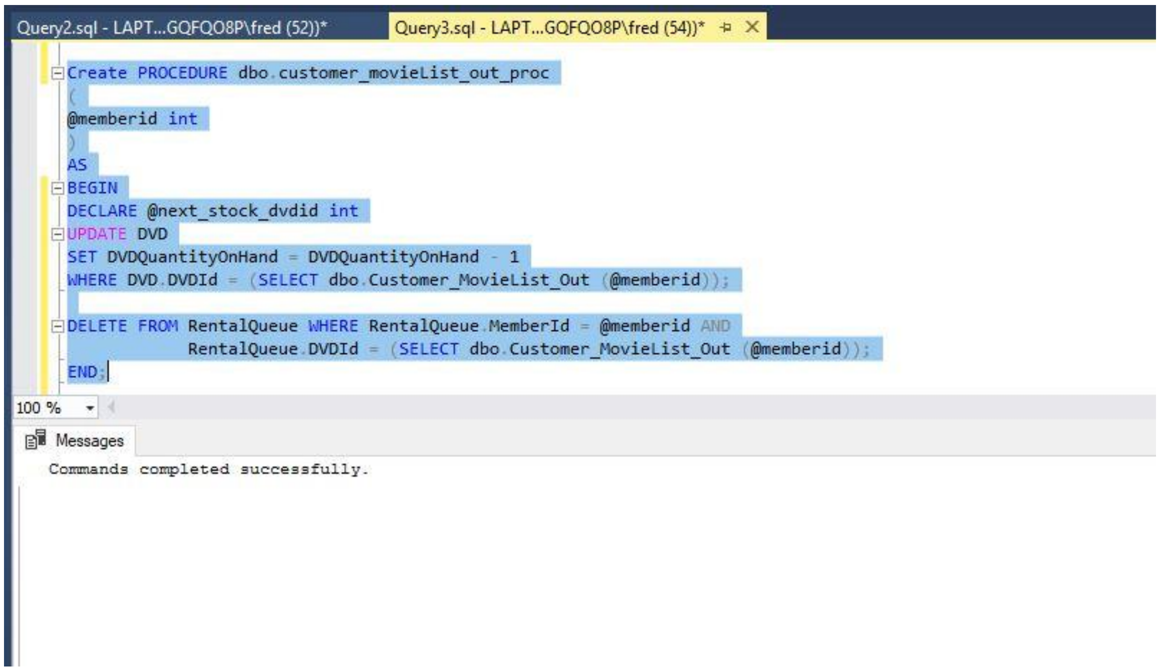
b. **SQL Server Note**: because functions in SQL Server cannot affect the state of the database which means we cannot perform insert, delete, update on the database (it can on variables), you will have to implement this as a stored procedure, but you are required to implement part of this functionality as a function which is called by the stored procedure.

```sql
Create FUNCTION dbo.Customer_MovieList_Out
(
 @MemberId int
)
RETURNS int
AS
BEGIN
-- Declare variables
       DECLARE @Next_Stock_DVDId int
          DECLARE @Member_Account int
          DECLARE @error NVARCHAR(MAX)
SET @Next_Stock_DVDId = (SELECT RentalQueue.DVDId FROM RentalQueue JOIN
Member ON RentalQueue.MemberId = Member.MemberId WHERE Member.MemberAccount > 0 AND
RentalQueue.MemberId = @MemberId AND RentalQueue.Position = 1 AND RentalQueue.DVDId IN
(SELECT DVD.DVDId FROM DVD WHERE DVD.DVDQuantityOnHand > 0));
SET @Member_Account = (SELECT Member.MemberAccount FROM Member WHERE MemberId =
@MemberId);

IF @Member_Account <= 0
BEGIN
SET @error = ('Customer balance cannot be zero or negative')
RETURN @error
END
IF @Next_Stock_DVDId = ''
   BEGIN
   RETURN NULL
   END
RETURN  @Next_Stock_DVDId
END;
```



```
Query2.sql - LAPT...GQFQO8P\fred (52))*    Query3.sql - LAPT...GQFQO8P\fred (54))*
Create FUNCTION dbo.Customer_MovieList_Out
(
 @MemberId int
)
RETURNS int
AS
BEGIN
-- Declare variables
       DECLARE @Next_Stock_DVDId int
       DECLARE @Member_Account int
       DECLARE @error NVARCHAR(MAX)
SET @Next_Stock_DVDId = (SELECT RentalQueue.DVDId FROM RentalQueue JOIN
Member ON RentalQueue.MemberId = Member.MemberId WHERE Member.MemberAccount > 0 AND
RentalQueue.MemberId = @MemberId AND RentalQueue.Position = 1 AND RentalQueue.DVDId IN
(SELECT DVD.DVDId FROM DVD WHERE DVD.DVDQuantityOnHand > 0));
SET @Member_Account = (SELECT Member.MemberAccount FROM Member WHERE MemberId = @MemberId);

IF @Member_Account <= 0
BEGIN
SET @error = ('Customer balance cannot be zero or negative')
RETURN @error
END
IF @Next_Stock_DVDId = ''
   BEGIN
   RETURN NULL
   END
RETURN  @Next_Stock_DVDId
END;
100 %
Messages
 Commands completed successfully.
100 %
Query executed successfully.    LAPTOP-MGQFQO8P (13.0 SP1) | LAPTOP-MGQFQO8P\fred (54) | NetFlix
```

```
Create PROCEDURE dbo.customer_movieList_out_proc
(
@memberid int
)
AS
BEGIN
DECLARE @next_stock_dvdid int
UPDATE DVD
SET DVDQuantityOnHand = DVDQuantityOnHand - 1
WHERE DVD.DVDId = (SELECT dbo.Customer_MovieList_Out (@memberid));

DELETE FROM RentalQueue WHERE RentalQueue.MemberId = @memberid AND
            RentalQueue.DVDId = (SELECT dbo.Customer_MovieList_Out (@memberid));
END;
```



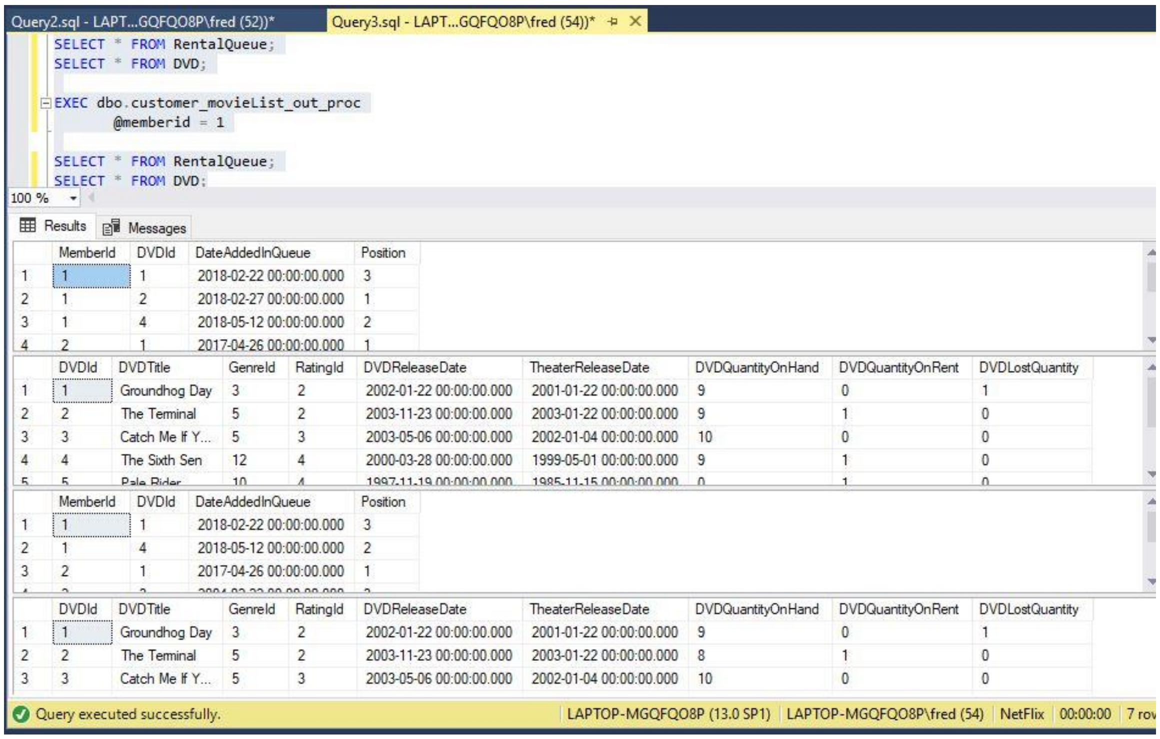
Query2.sql - LAPT...GQFQO8P\fred (52))* | Query3.sql - LAPT...GQFQO8P\fred (54))*

```sql
Create PROCEDURE dbo.customer_movieList_out_proc
(
@memberid int
)
AS
BEGIN
DECLARE @next_stock_dvdid int
UPDATE DVD
SET DVDQuantityOnHand = DVDQuantityOnHand - 1
WHERE DVD.DVDId = (SELECT dbo.Customer_MovieList_Out (@memberid));

DELETE FROM RentalQueue WHERE RentalQueue.MemberId = @memberid AND
            RentalQueue.DVDId = (SELECT dbo.Customer_MovieList_Out (@memberid));
END;
```

100 %

Messages

```
Commands completed successfully.
```



Query2.sql - LAPT...GQFQO8P\fred (52))* | Query3.sql - LAPT...GQFQO8P\fred (54))*

```sql
SELECT * FROM RentalQueue;
SELECT * FROM DVD;

EXEC dbo.customer_movieList_out_proc
        @memberid = 1

SELECT * FROM RentalQueue;
SELECT * FROM DVD;
```

100 %

Results | Messages

	MemberId	DVDId	DateAddedInQueue	Position
1	1	1	2018-02-22 00:00:00.000	3
2	1	2	2018-02-27 00:00:00.000	1
3	1	4	2018-05-12 00:00:00.000	2
4	2	1	2017-04-26 00:00:00.000	1

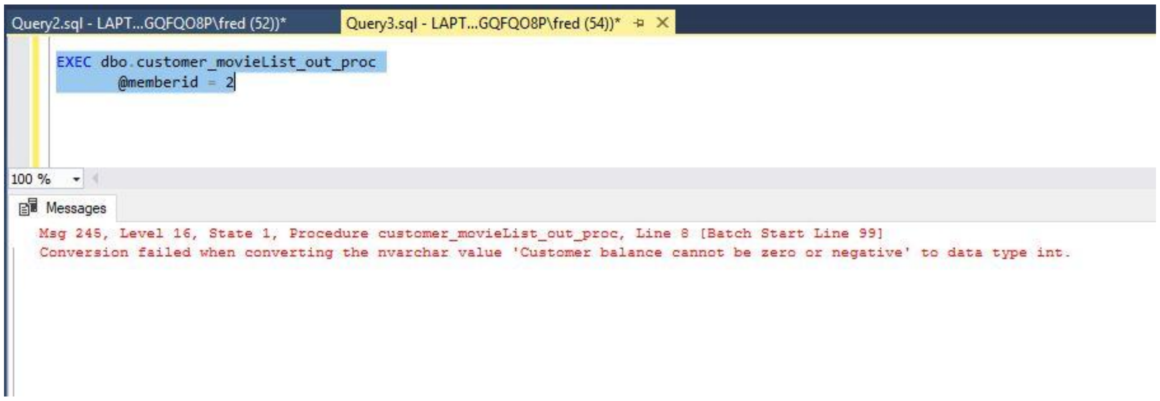
	DVDId	DVDTitle	GenreId	RatingId	DVDReleaseDate	TheaterReleaseDate	DVDQuantityOnHand	DVDQuantityOnRent	DVDLostQuantity
1	1	Groundhog Day	3	2	2002-01-22 00:00:00.000	2001-01-22 00:00:00.000	9	0	1
2	2	The Terminal	5	2	2003-11-23 00:00:00.000	2003-01-22 00:00:00.000	9	1	0
3	3	Catch Me If Y...	5	3	2003-05-06 00:00:00.000	2002-01-04 00:00:00.000	10	0	0
4	4	The Sixth Sen	12	4	2000-03-28 00:00:00.000	1999-05-01 00:00:00.000	9	1	0
5	5	Pale Rider	10	4	1997-11-19 00:00:00.000	1985-11-15 00:00:00.000	0	1	0

	MemberId	DVDId	DateAddedInQueue	Position
1	1	1	2018-02-22 00:00:00.000	3
2	1	4	2018-05-12 00:00:00.000	2
3	2	1	2017-04-26 00:00:00.000	1

	DVDId	DVDTitle	GenreId	RatingId	DVDReleaseDate	TheaterReleaseDate	DVDQuantityOnHand	DVDQuantityOnRent	DVDLostQuantity
1	1	Groundhog Day	3	2	2002-01-22 00:00:00.000	2001-01-22 00:00:00.000	9	0	1
2	2	The Terminal	5	2	2003-11-23 00:00:00.000	2003-01-22 00:00:00.000	8	1	0
3	3	Catch Me If Y...	5	3	2003-05-06 00:00:00.000	2002-01-04 00:00:00.000	10	0	0

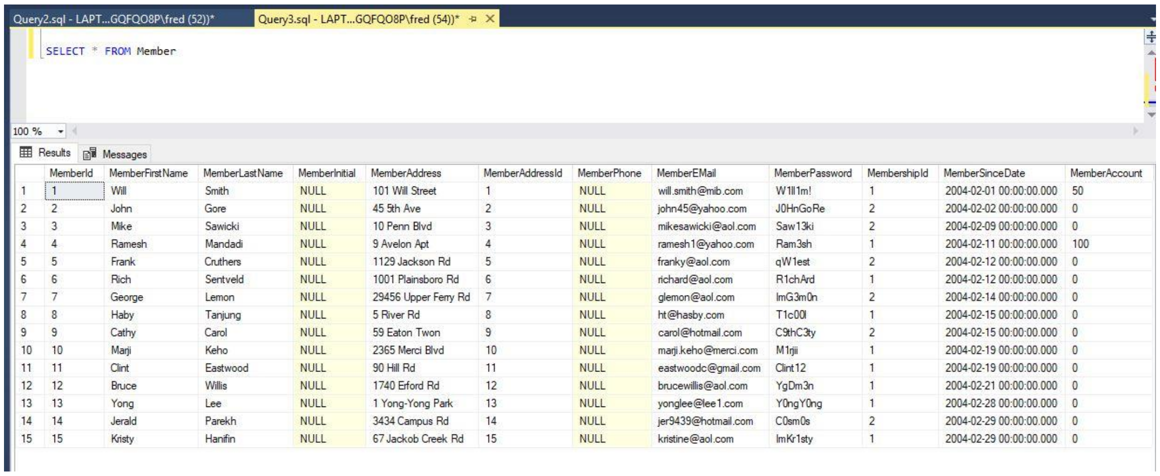
Query executed successfully. | LAPTOP-MGQFQO8P (13.0 SP1) | LAPTOP-MGQFQO8P\fred (54) | NetFlix | 00:00:00 | 7 ro

Additionally the function should perform error handling to check that the customer's account is valid for this operation.



```
EXEC dbo.customer_movieList_out_proc
        @memberid = 2
```

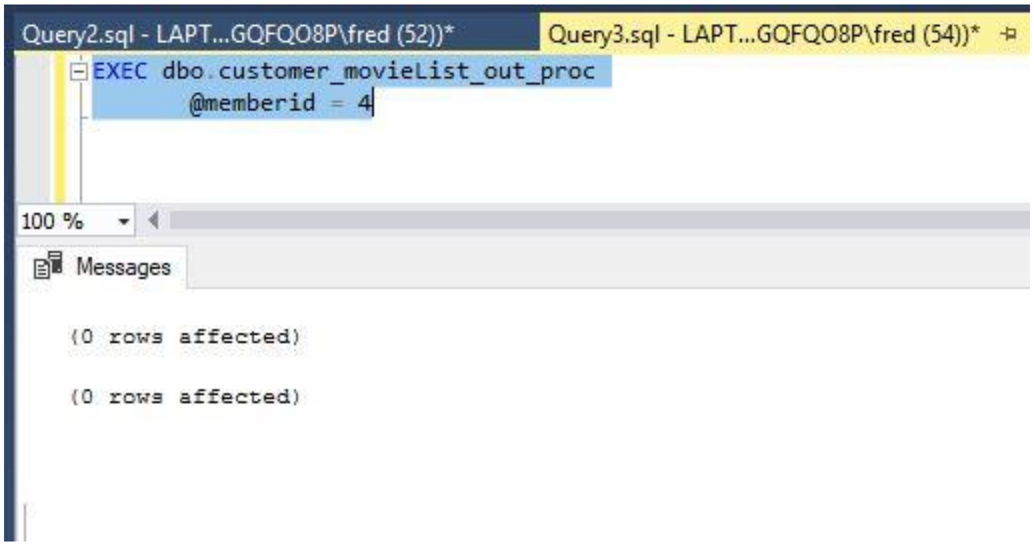
```
Msg 245, Level 16, State 1, Procedure customer_movieList_out_proc, Line 8 [Batch Start Line 99]
Conversion failed when converting the nvarchar value 'Customer balance cannot be zero or negative' to data type int.
```



```
SELECT * FROM Member
```

	MemberId	MemberFirstName	MemberLastName	MemberInitial	MemberAddress	MemberAddressId	MemberPhone	MemberEMail	MemberPassword	MembershipId	MemberSinceDate	MemberAccount
1	1	Will	Smith	NULL	101 Will Street	1	NULL	will.smith@mib.com	W1ll1m!	1	2004-02-01 00:00:00.000	50
2	2	John	Gore	NULL	45 5th Ave	2	NULL	john45@yahoo.com	J0HnGoRe	2	2004-02-02 00:00:00.000	0
3	3	Mike	Sawicki	NULL	10 Penn Blvd	3	NULL	mikesawicki@aol.com	Saw13ki	2	2004-02-09 00:00:00.000	0
4	4	Ramesh	Mandadi	NULL	9 Avelon Apt	4	NULL	ramesh1@yahoo.com	Ram3sh	1	2004-02-11 00:00:00.000	100
5	5	Frank	Cruthers	NULL	1129 Jackson Rd	5	NULL	franky@aol.com	qW1est	2	2004-02-12 00:00:00.000	0
6	6	Rich	Sentveld	NULL	1001 Plainsboro Rd	6	NULL	richard@aol.com	R1chArd	1	2004-02-12 00:00:00.000	0
7	7	George	Lemon	NULL	29456 Upper Ferry Rd	7	NULL	glemon@aol.com	lmG3m0n	2	2004-02-14 00:00:00.000	0
8	8	Haby	Tanjung	NULL	5 River Rd	8	NULL	ht@hasby.com	T1c00l	1	2004-02-15 00:00:00.000	0
9	9	Cathy	Carol	NULL	59 Eaton Twon	9	NULL	carol@hotmail.com	C9thC3ty	2	2004-02-15 00:00:00.000	0
10	10	Marji	Keho	NULL	2365 Merci Blvd	10	NULL	marji.keho@merci.com	M1rjii	1	2004-02-19 00:00:00.000	0
11	11	Clint	Eastwood	NULL	90 Hill Rd	11	NULL	eastwoodc@gmail.com	Clint12	1	2004-02-19 00:00:00.000	0
12	12	Bruce	Willis	NULL	1740 Erford Rd	12	NULL	brucewillis@aol.com	YgDm3n	1	2004-02-21 00:00:00.000	0
13	13	Yong	Lee	NULL	1 Yong-Yong Park	13	NULL	yonglee@lee1.com	Y0ngY0ng	1	2004-02-28 00:00:00.000	0
14	14	Jerald	Parekh	NULL	3434 Campus Rd	14	NULL	jer9439@hotmail.com	C0sm0s	2	2004-02-29 00:00:00.000	0
15	15	Kristy	Hanifin	NULL	67 Jackob Creek Rd	15	NULL	kristine@aol.com	lmKr1sty	1	2004-02-29 00:00:00.000	0

If the movie list is empty or none of the titles are in stock, then the function should return NULL.

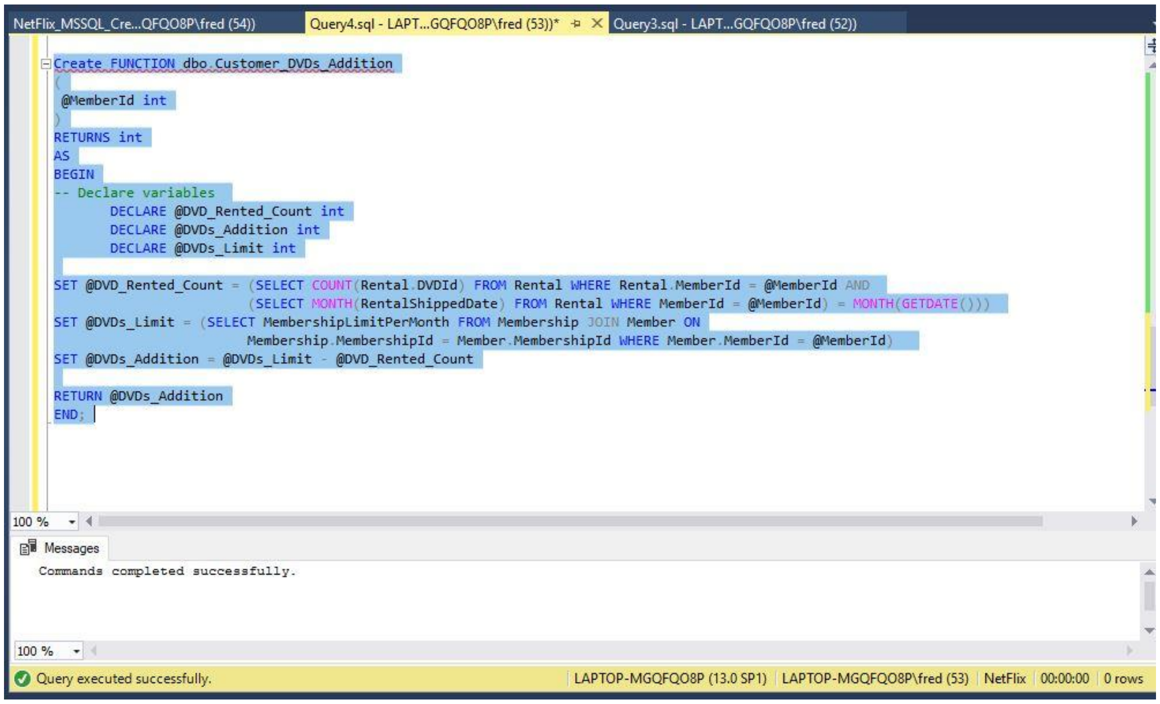


```
EXEC dbo.customer_movieList_out_proc
        @memberid = 4
```

```
(0 rows affected)

(0 rows affected)
```

2. Write a **function** that returns the number of additional DVDs that a customer may receive before they reach the limits of their contract. The function should take the customer ID as an IN parameter and should return the number of DVDs through its integer function value.

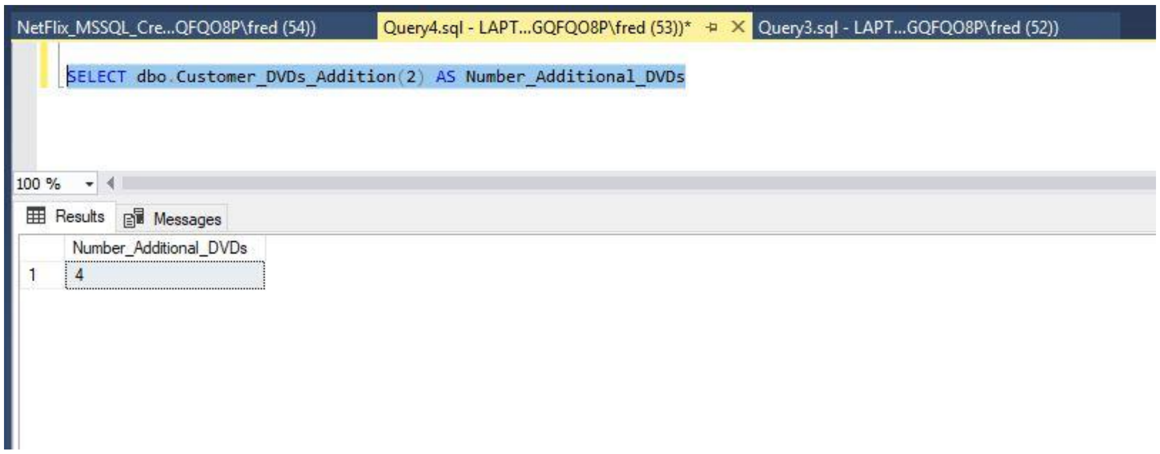
```
Create FUNCTION dbo.Customer_DVDs_Addition
(
 @MemberId int
)
RETURNS int
AS
BEGIN
-- Declare variables
       DECLARE @DVD_Rented_Count int
       DECLARE @DVDs_Addition int
       DECLARE @DVDs_Limit int

SET @DVD_Rented_Count = (SELECT COUNT(Rental.DVDId) FROM Rental WHERE
                         Rental.MemberId = @MemberId AND
                         (SELECT MONTH(RentalShippedDate) FROM Rental WHERE
                         MemberId = @MemberId) = MONTH(GETDATE()))

SET @DVDs_Limit = (SELECT MembershipLimitPerMonth FROM Membership JOIN Member ON
                        Membership.MembershipId = Member.MembershipId WHERE
                        Member.MemberId = @MemberId)

SET @DVDs_Addition = @DVDs_Limit - @DVD_Rented_Count

RETURN @DVDs_Addition
END;
```



```
Create FUNCTION dbo.Customer_DVDs_Addition
(
 @MemberId int
)
RETURNS int
AS
BEGIN
-- Declare variables
       DECLARE @DVD_Rented_Count int
       DECLARE @DVDs_Addition int
       DECLARE @DVDs_Limit int

SET @DVD_Rented_Count = (SELECT COUNT(Rental.DVDId) FROM Rental WHERE Rental.MemberId = @MemberId AND
                        (SELECT MONTH(RentalShippedDate) FROM Rental WHERE MemberId = @MemberId) = MONTH(GETDATE()))
SET @DVDs_Limit = (SELECT MembershipLimitPerMonth FROM Membership JOIN Member ON
                        Membership.MembershipId = Member.MembershipId WHERE Member.MemberId = @MemberId)
SET @DVDs_Addition = @DVDs_Limit - @DVD_Rented_Count

RETURN @DVDs_Addition
END;
```

```
Commands completed successfully.
```

```
Query executed successfully.   LAPTOP-MGQFQO8P (13.0 SP1) | LAPTOP-MGQFQO8P\fred (53) | NetFlix | 00:00:00 | 0 rows
```



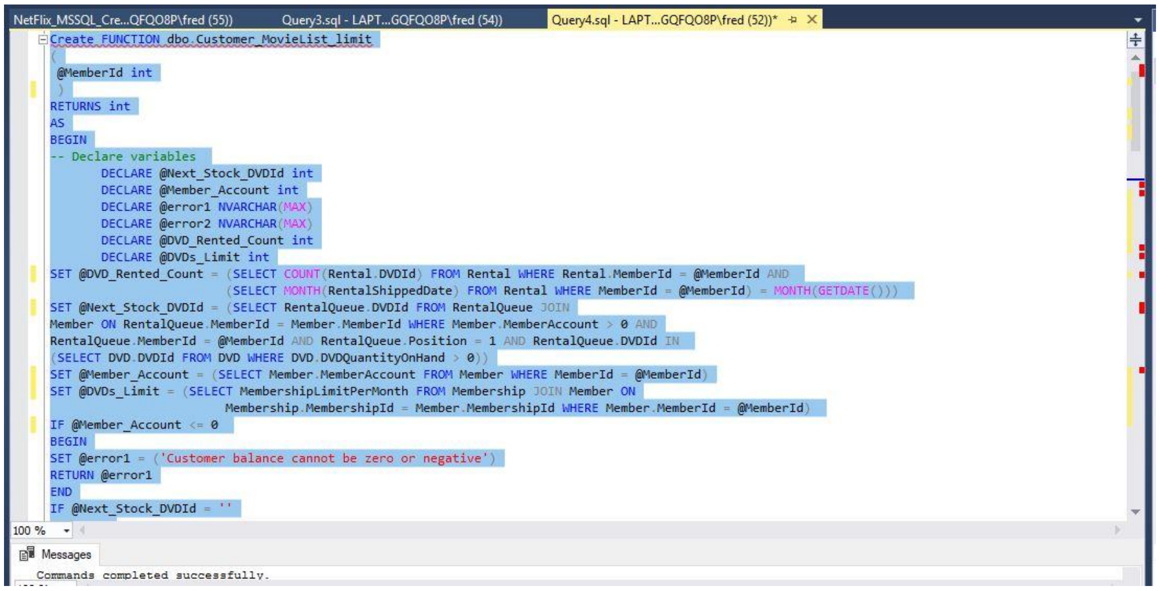
```
SELECT dbo.Customer_DVDs_Addition(2) AS Number_Additional_DVDs
```

	Number_Additional_DVDs
1	4

3. Write a **stored procedure** that implements the processing when a DVD is returned in the mail from a customer and the next DVD is sent out. This processing should include recording that the DVD has been returned and should also determine the number of additional DVDs that should be mailed to the customer. **Use the functions and stored procedures which you have already created to complete this transaction**. It might be helpful to write out the steps of your logic first, for example (this may not have all the steps required, but is an example):
   a. Customer returns a DVD
   b. Initiate the function from question 2 to return the number of additional dvds which can be rented.

c. Initiate the function/stored procedure from question 1 to get a movie from the customer's request list (rentalqueue) which is in stock.
d. Perform the rental of the above DVD

```
Create FUNCTION dbo.Customer_MovieList_limit
(
 @MemberId int
 )
RETURNS int
AS
BEGIN
-- Declare variables
       DECLARE @Next_Stock_DVDId int
          DECLARE @Member_Account int
          DECLARE @error1 NVARCHAR(MAX)
          DECLARE @error2 NVARCHAR(MAX)
          DECLARE @DVD_Rented_Count int
          DECLARE @DVDs_Limit int
SET @DVD_Rented_Count = (SELECT COUNT(Rental.DVDId) FROM Rental WHERE Rental.MemberId =
@MemberId AND
                           (SELECT MONTH(RentalShippedDate) FROM Rental WHERE MemberId =
@MemberId) = MONTH(GETDATE()))
SET @Next_Stock_DVDId = (SELECT RentalQueue.DVDId FROM RentalQueue JOIN
Member ON RentalQueue.MemberId = Member.MemberId WHERE Member.MemberAccount > 0 AND
RentalQueue.MemberId = @MemberId AND RentalQueue.Position = 1 AND RentalQueue.DVDId IN
(SELECT DVD.DVDId FROM DVD WHERE DVD.DVDQuantityOnHand > 0))
SET @Member_Account = (SELECT Member.MemberAccount FROM Member WHERE MemberId =
@MemberId)
SET @DVDs_Limit = (SELECT MembershipLimitPerMonth FROM Membership JOIN Member ON
                        Membership.MembershipId = Member.MembershipId WHERE
Member.MemberId = @MemberId)
IF @Member_Account <= 0
BEGIN
SET @error1 = ('Customer balance cannot be zero or negative')
RETURN @error1
END
IF @Next_Stock_DVDId = ''
   BEGIN
   RETURN NULL
   END
IF @DVD_Rented_Count >= @DVDs_Limit
BEGIN
SET @error2 = ('The number of month_dvds requested is more than the limit of your
contract')
RETURN @error2
END
RETURN  @Next_Stock_DVDId
END;
```



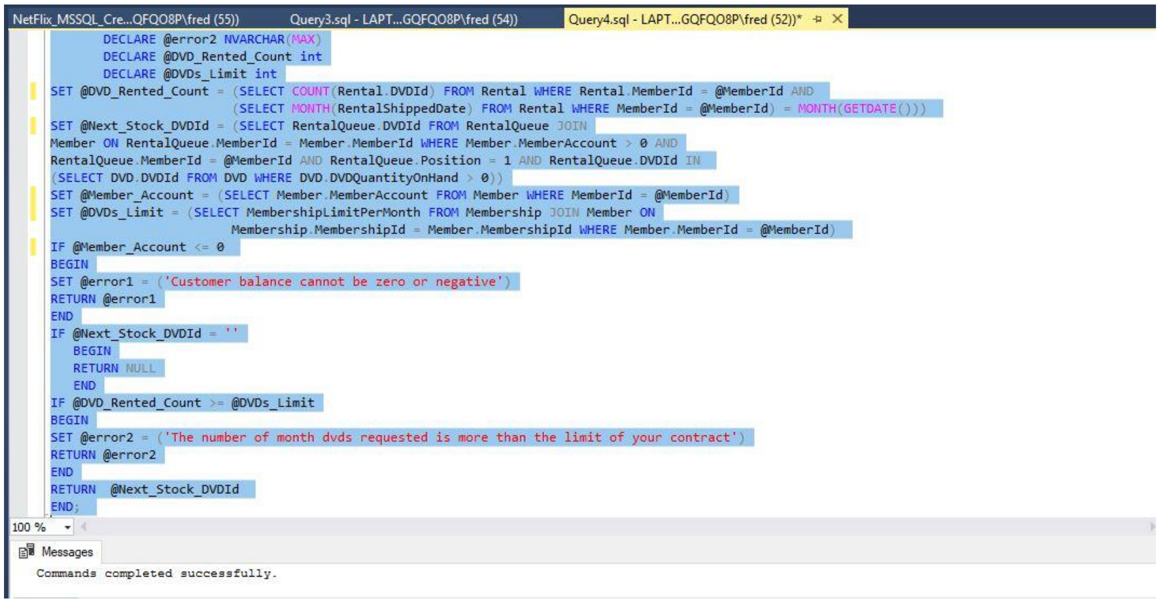
NetFlix_MSSQL_Cre...QFQO8P\fred (55))   Query3.sql - LAPT...GQFQO8P\fred (54))   Query4.sql - LAPT...GQFQO8P\fred (52))*

```sql
Create FUNCTION dbo.Customer_MovieList_limit
(
 @MemberId int
 )
RETURNS int
AS
BEGIN
-- Declare variables
       DECLARE @Next_Stock_DVDId int
       DECLARE @Member_Account int
       DECLARE @error1 NVARCHAR(MAX)
       DECLARE @error2 NVARCHAR(MAX)
       DECLARE @DVD_Rented_Count int
       DECLARE @DVDs_Limit int
SET @DVD_Rented_Count = (SELECT COUNT(Rental.DVDId) FROM Rental WHERE Rental.MemberId = @MemberId AND
                        (SELECT MONTH(RentalShippedDate) FROM Rental WHERE MemberId = @MemberId) = MONTH(GETDATE()))
SET @Next_Stock_DVDId = (SELECT RentalQueue.DVDId FROM RentalQueue JOIN
Member ON RentalQueue.MemberId = Member.MemberId WHERE Member.MemberAccount > 0 AND
RentalQueue.MemberId = @MemberId AND RentalQueue.Position = 1 AND RentalQueue.DVDId IN
(SELECT DVD.DVDId FROM DVD WHERE DVD.DVDQuantityOnHand > 0))
SET @Member_Account = (SELECT Member.MemberAccount FROM Member WHERE MemberId = @MemberId)
SET @DVDs_Limit = (SELECT MembershipLimitPerMonth FROM Membership JOIN Member ON
                        Membership.MembershipId = Member.MembershipId WHERE Member.MemberId = @MemberId)
IF @Member_Account <= 0
BEGIN
SET @error1 = ('Customer balance cannot be zero or negative')
RETURN @error1
END
IF @Next_Stock_DVDId = ''
```

100 %

Messages

```
Commands completed successfully.
```



NetFlix_MSSQL_Cre...QFQO8P\fred (55))   Query3.sql - LAPT...GQFQO8P\fred (54))   Query4.sql - LAPT...GQFQO8P\fred (52))*

```sql
       DECLARE @error2 NVARCHAR(MAX)
       DECLARE @DVD_Rented_Count int
       DECLARE @DVDs_Limit int
SET @DVD_Rented_Count = (SELECT COUNT(Rental.DVDId) FROM Rental WHERE Rental.MemberId = @MemberId AND
                        (SELECT MONTH(RentalShippedDate) FROM Rental WHERE MemberId = @MemberId) = MONTH(GETDATE()))
SET @Next_Stock_DVDId = (SELECT RentalQueue.DVDId FROM RentalQueue JOIN
Member ON RentalQueue.MemberId = Member.MemberId WHERE Member.MemberAccount > 0 AND
RentalQueue.MemberId = @MemberId AND RentalQueue.Position = 1 AND RentalQueue.DVDId IN
(SELECT DVD.DVDId FROM DVD WHERE DVD.DVDQuantityOnHand > 0))
SET @Member_Account = (SELECT Member.MemberAccount FROM Member WHERE MemberId = @MemberId)
SET @DVDs_Limit = (SELECT MembershipLimitPerMonth FROM Membership JOIN Member ON
                        Membership.MembershipId = Member.MembershipId WHERE Member.MemberId = @MemberId)
IF @Member_Account <= 0
BEGIN
SET @error1 = ('Customer balance cannot be zero or negative')
RETURN @error1
END
IF @Next_Stock_DVDId = ''
   BEGIN
   RETURN NULL
   END
IF @DVD_Rented_Count >= @DVDs_Limit
BEGIN
SET @error2 = ('The number of month dvds requested is more than the limit of your contract')
RETURN @error2
END
RETURN  @Next_Stock_DVDId
END;
```

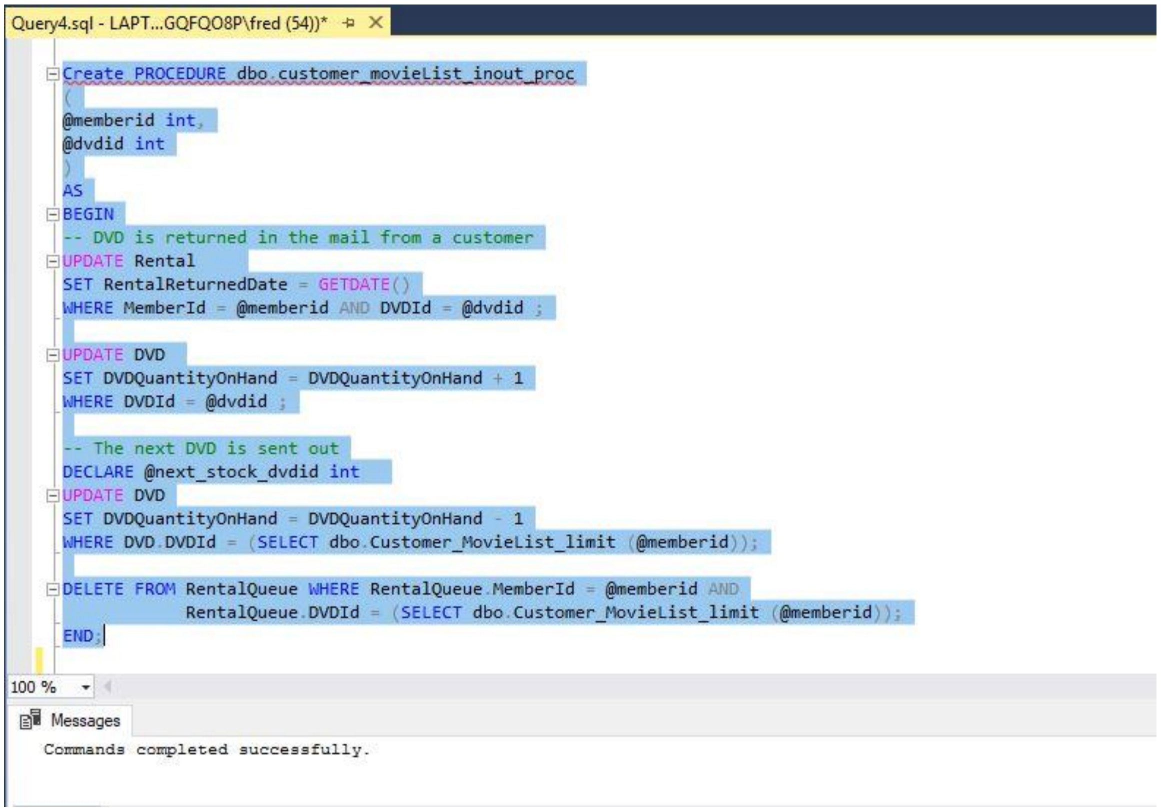
100 %

Messages

```
Commands completed successfully.
```

```sql
Create PROCEDURE dbo.customer_movieList_inout_proc
(
@memberid int,
@dvdid int
)
AS
BEGIN
-- DVD is returned in the mail from a customer
UPDATE Rental
SET RentalReturnedDate = GETDATE()
WHERE MemberId = @memberid AND DVDId = @dvdid ;
```

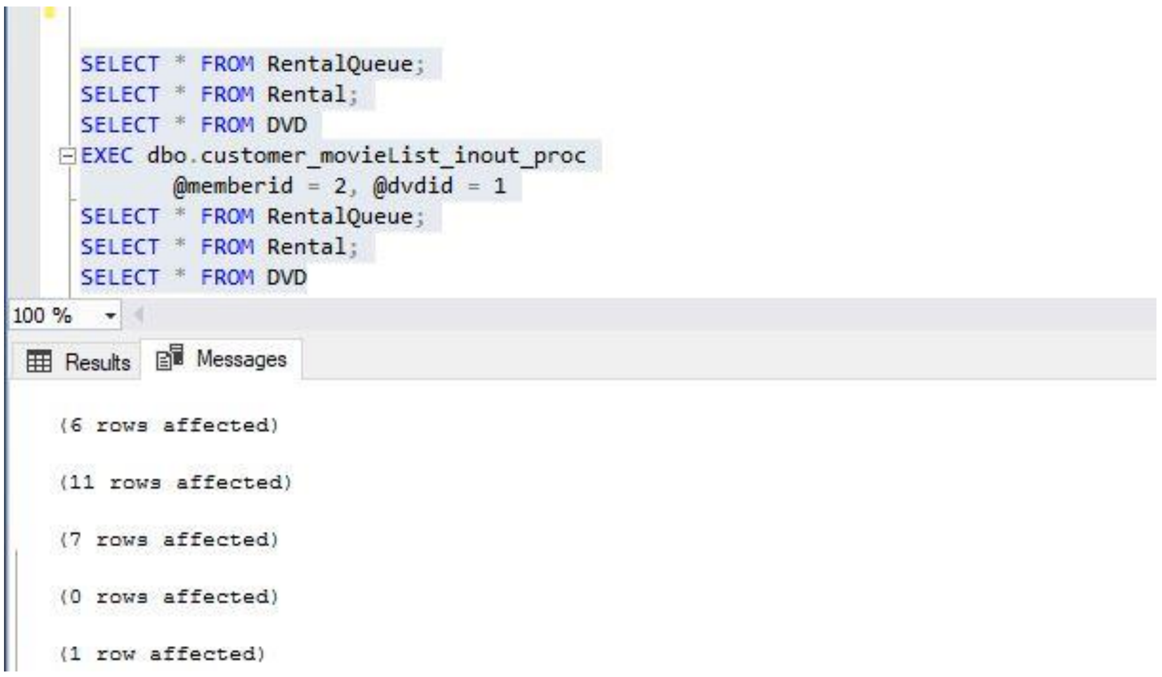
```
UPDATE DVD
SET DVDQuantityOnHand = DVDQuantityOnHand + 1
WHERE DVDId = @dvdid ;

-- The next DVD is sent out
DECLARE @next_stock_dvdid int
UPDATE DVD
SET DVDQuantityOnHand = DVDQuantityOnHand - 1
WHERE DVD.DVDId = (SELECT dbo.Customer_MovieList_limit (@memberid));

DELETE FROM RentalQueue WHERE RentalQueue.MemberId = @memberid AND
            RentalQueue.DVDId = (SELECT dbo.Customer_MovieList_limit (@memberid));
END;
```



Query4.sql - LAPT...GQFQO8P\fred (54))*

```
Create PROCEDURE dbo.customer_movieList_inout_proc
(
@memberid int,
@dvdid int
)
AS
BEGIN
-- DVD is returned in the mail from a customer
UPDATE Rental
SET RentalReturnedDate = GETDATE()
WHERE MemberId = @memberid AND DVDId = @dvdid ;

UPDATE DVD
SET DVDQuantityOnHand = DVDQuantityOnHand + 1
WHERE DVDId = @dvdid ;

-- The next DVD is sent out
DECLARE @next_stock_dvdid int
UPDATE DVD
SET DVDQuantityOnHand = DVDQuantityOnHand - 1
WHERE DVD.DVDId = (SELECT dbo.Customer_MovieList_limit (@memberid));

DELETE FROM RentalQueue WHERE RentalQueue.MemberId = @memberid AND
            RentalQueue.DVDId = (SELECT dbo.Customer_MovieList_limit (@memberid));
END;
```

```
Commands completed successfully.
```



```
SELECT * FROM RentalQueue;
SELECT * FROM Rental;
SELECT * FROM DVD
EXEC dbo.customer_movieList_inout_proc
       @memberid = 2, @dvdid = 1
SELECT * FROM RentalQueue;
SELECT * FROM Rental;
SELECT * FROM DVD
```

```
(6 rows affected)

(11 rows affected)

(7 rows affected)

(0 rows affected)

(1 row affected)
```