Use this document to capture the screenshots requested in the instructions for assignment 3.0.

**Name**: Mi Gao

**Date: 08/01/2020**

**The submission template uses the following format :**

* Document question number and your code describing what each query is doing.
* Paste in your code.
* Paste in a screenshot of the results running your code.

1. **The function** that returns the DVD ID of the next in stock DVD in the customer’s movie list (rental queue). In here, the next one is based on the time which is added first. **Error handling with two conditions which the things in the queue or the quantity of that DVD. Otherwise, return that DVD id.**

CREATE OR REPLACE FUNCTION public.func\_return\_firstone\_in\_rentalqueue(member\_id integer)

RETURNS numeric

LANGUAGE 'plpgsql'

VOLATILE

PARALLEL UNSAFE

COST 100

AS $BODY$declare

dvd\_id integer;

begin

-- find dvdid by member\_id input

select rentalqueue.dvdid

into dvd\_id

from rentalqueue

where memberid = member\_id

ORDER BY dateaddedinqueue ASC FETCH FIRST ROW ONLY; -- first dvd added to his queue based on the date

if not found then

raise 'Error, nothing in the rental queue of this customer yet.';

elseif (select dvdquantityonhand from dvd where dvd.dvdid = dvd\_id)=0 then

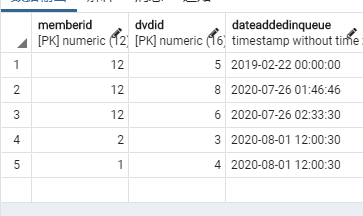
raise 'Error, none of this dvd is in stock.'; -- the quantity is out of stock which is ZERO

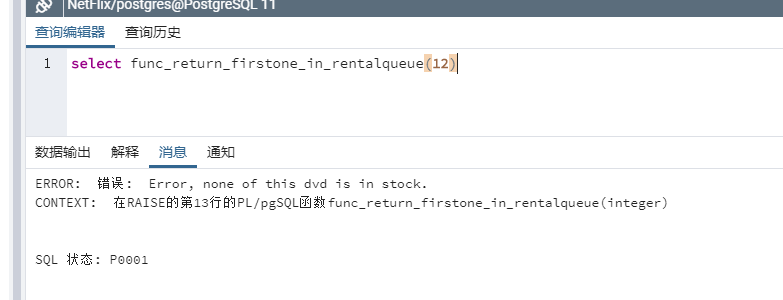
else return dvd\_id;

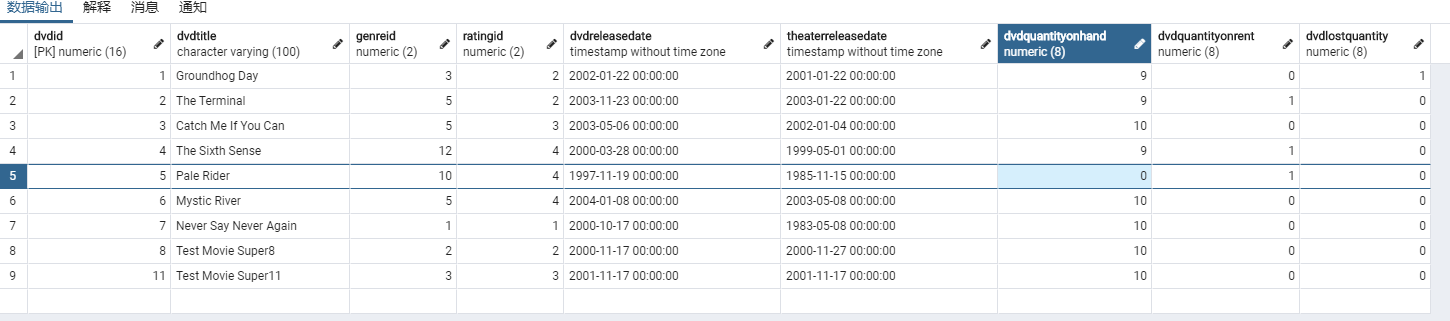
end if;

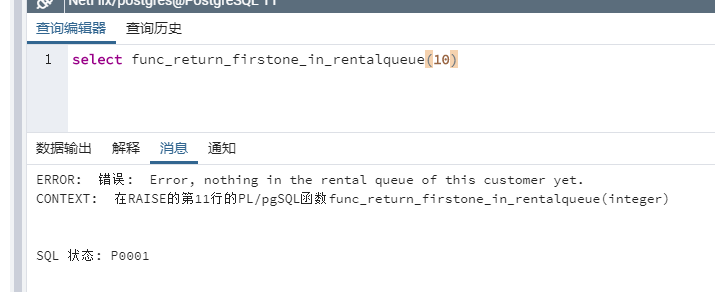
return NULL;

end;$BODY$;











1. **I do not quite understand the meaning of “**the number of additional DVDs that a customer may receive before they reach the limits of their contract**” even after you explained to me in the class. However in my understanding, this question ask me to show the number of rental I left in same-time so that why it needs to be limited by those two conditions. If it only want me show how many of the total number of rental left in this month, it looks too easy by one line query and does not need those comparisons. Thus, this is how I do it:**

CREATE OR REPLACE FUNCTION public.func\_return\_remainingrentalonetime(member\_id integer)

RETURNS integer

LANGUAGE 'plpgsql'

VOLATILE

PARALLEL UNSAFE

COST 100

AS $BODY$

declare

membershiptype integer;

cur\_dvdcanrent integer;

cur\_dvdholding integer;

num\_monthlimit integer;

num\_rentalleft integer;

begin

SELECT member.membershipid -- check membership type

into membershiptype

from membership

inner join member on membership.membershipid = member.membershipid

where memberid = member\_id;

-- Counting how much holding in hands totally

cur\_dvdholding :=

(select count(rentalid)

from rental

inner join member on rental.memberid = member.memberid

where rentalreturneddate is null and rental.memberid = member\_id);

-- based on membership type then set the default number of DVD can rent now

if (membershiptype = 1) then

cur\_dvdcanrent := 3;

num\_monthlimit := 99;

-- has 3 at-one-time now

if (cur\_dvdholding = 3) then -- 3 copies not returned yet

cur\_dvdcanrent := 0; -- return 0

return cur\_dvdcanrent;

end if;

elseif (membershiptype = 2) then

cur\_dvdcanrent := 2;

num\_monthlimit := 4;

-- has 2 at-one-time now

if (cur\_dvdholding = 2) then -- 2 copies not returned yet

cur\_dvdcanrent := 0; -- return 0

return cur\_dvdcanrent;

end if;

end if;

-- Counting how much left in this month

num\_rentalleft := (num\_monthlimit - (select count(rentalid)

from rental

inner join member on rental.memberid = member.memberid

where EXTRACT(MONTH FROM rentalrequestdate) = EXTRACT(MONTH FROM current\_date)

and EXTRACT(YEAR FROM rentalrequestdate) = EXTRACT(YEAR FROM current\_date)

and rental.memberid = member\_id));

if (num\_rentalleft = 0) then -- hit the limitation totally

cur\_dvdcanrent := num\_rentalleft; -- return 0

elseif (num\_rentalleft < 3) then -- only left at most 2 changes totally in this month

cur\_dvdcanrent := num\_rentalleft - cur\_dvdholding; -- 2 changes reduced by current holding 0 to 2 copies

if (cur\_dvdholding > num\_rentalleft) then -- But if already hold too many in hands

cur\_dvdcanrent := 0;

end if;

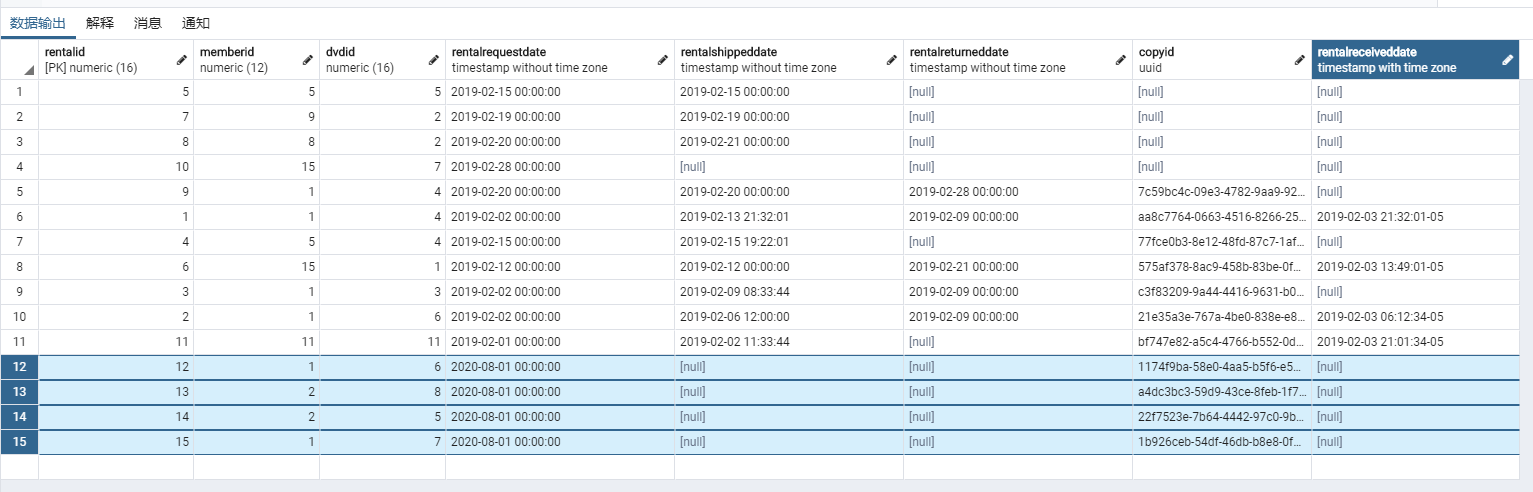
else cur\_dvdcanrent := cur\_dvdcanrent - cur\_dvdholding; -- basic counting

end if;

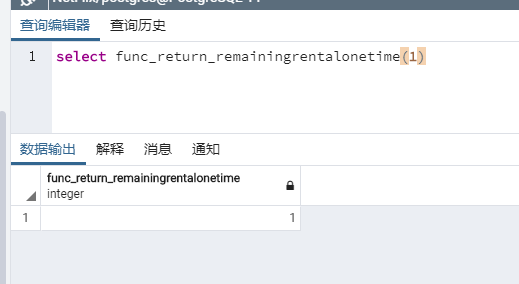
return cur\_dvdcanrent;

end;

$BODY$;



Here, I have member no.1 who own the plan A (3 in one time, 99 per month) has several rental history but only rent 2 times this month; and member no.2 with plan B (2 in one time, 4 per month) who only rent 2 times totally and did in this month.





1. **Before the insert tuple into rental table, check whether the customer reached the limitation or not (how many changes of the rental left).**

CREATE OR REPLACE FUNCTION public.trig\_func\_rentallimitcheck()

RETURNS trigger

LANGUAGE 'plpgsql'

VOLATILE

COST 100

AS $BODY$declare

member\_id integer; -- using to store the input of member id

result integer; -- using to store the output of the condition function

begin

IF TG\_OP = 'INSERT' THEN

member\_id := new.memberid; -- passing the new value of input

select func\_return\_remainingrentalonetime(member\_id) into result; -- computing through the function

if result = 0 then -- Now, if none of rental changes left

raise EXCEPTION 'Error, the monthly limitation has been reached.';

return old; -- rollback

end if;

end if;

return new;

end;

$BODY$;

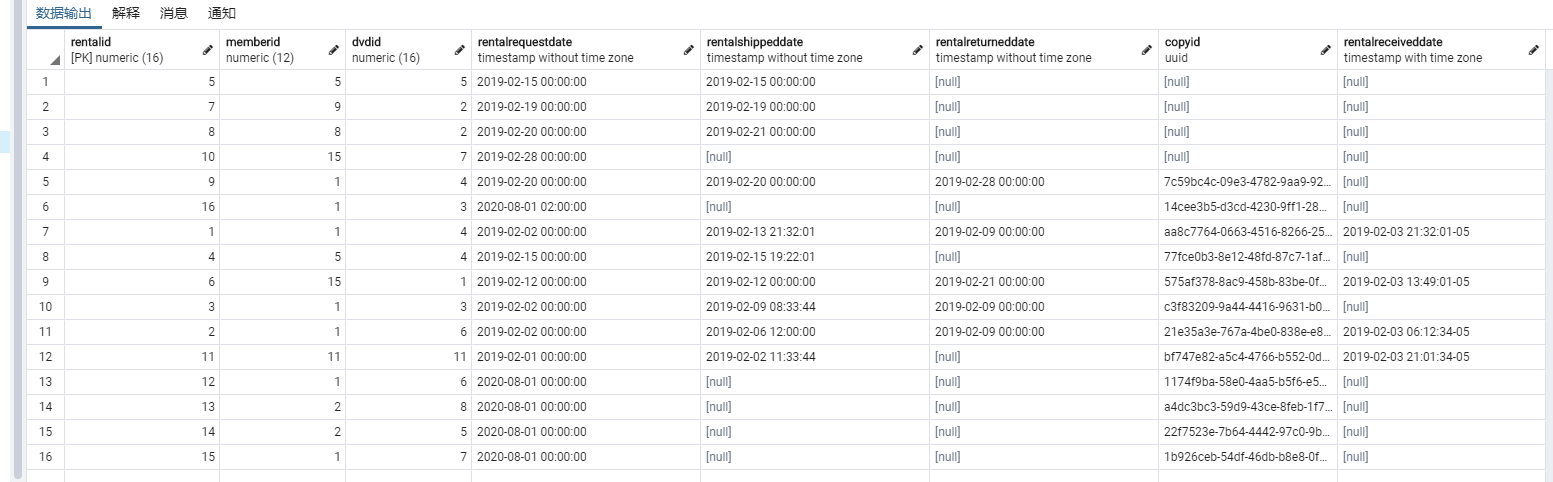
CREATE TRIGGER tri\_rentallimitcheck

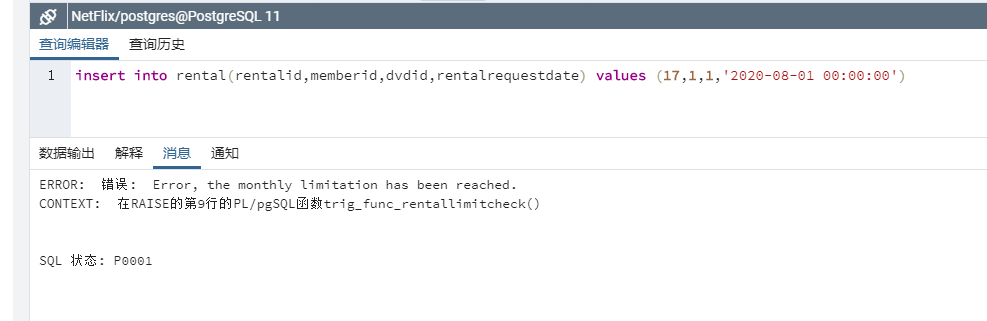
BEFORE INSERT -- please notice that before

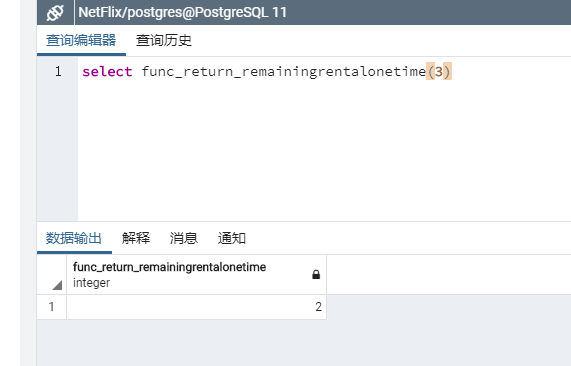
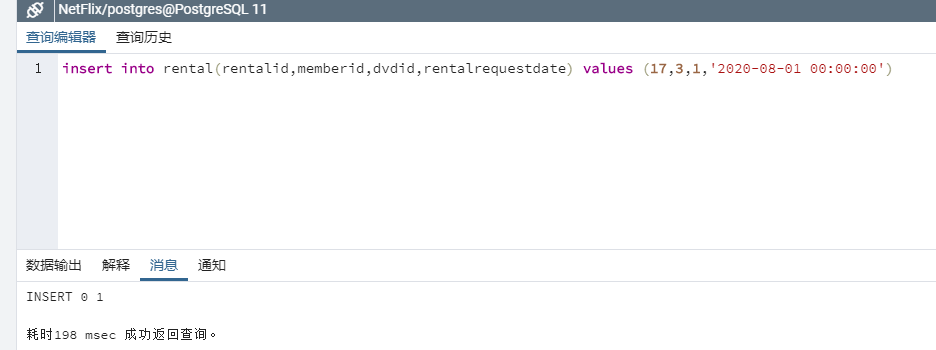
ON public.rental

FOR EACH ROW

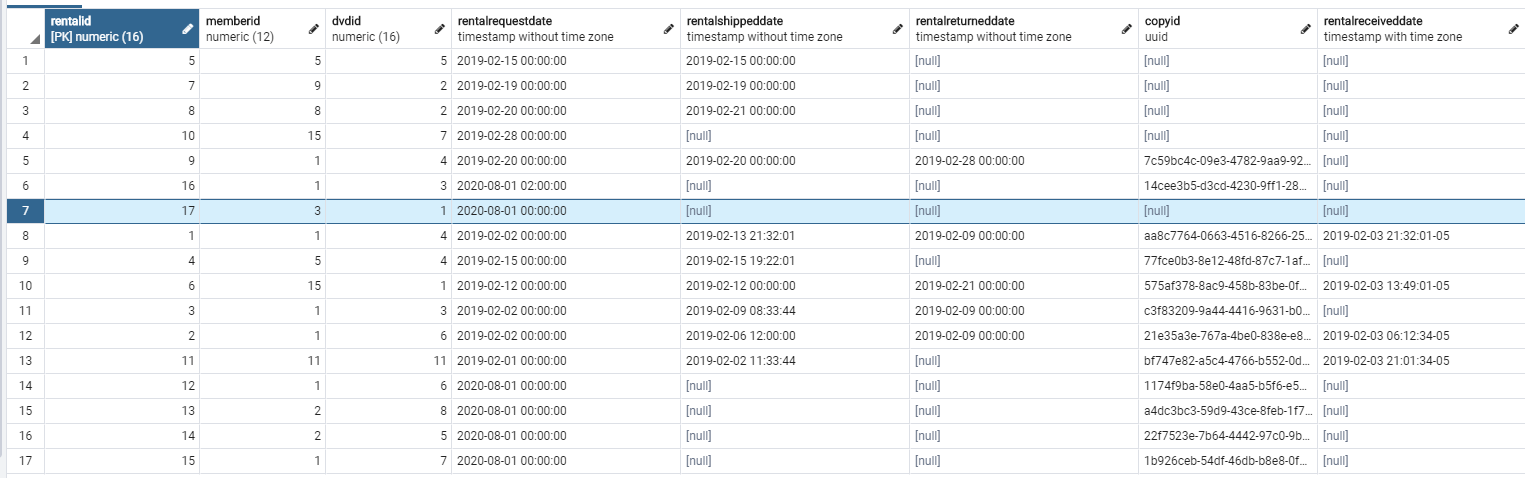
EXECUTE PROCEDURE public.trig\_func\_rentallimitcheck ();







1. This stored procedure has those features:
2. This processing include recording that the DVD has been returned and should also keep checking the number of additional DVDs that should be mailed to the customer.
3. Customer returns a DVD or notes the DVD is lost in which case they are charged against their account.
4. Initiate function from question 2 to return the number of additional dvds which can be rented.
5. Initiate the function from question 1 to get a movie from the customer’s request list (rentalqueue) which is in stock.
6. Initiate stored procedure from Advanced Programming part 2 which removes this DVD from the queue.
7. Perform the rental of the above DVDs.
8. Update all the DVD quantities accordingly.

-- apply auto increment sequence for payment first

ALTER TABLE payment ALTER COLUMN paymentid SET DEFAULT nextval('paymentid\_sequence')

CREATE OR REPLACE PROCEDURE public.rentalqueue\_processing(memberid\_arg integer, rentalid\_arg numeric, dvdid\_arg numeric, status\_arg boolean DEFAULT true, rentalreturneddate\_arg timestamp without time zone DEFAULT CURRENT\_TIMESTAMP(0))

LANGUAGE 'plpgsql'

AS $BODY$declare

paymentamount numeric(5,2); -- charges in lost

dvd\_id numeric; -- next dvd in rentalqueue

rentalchangesnow integer;

begin

paymentamount := (select membershipdvdlostprice from membership

inner join member on membership.membershipid = member.membershipid

where member.memberid = memberid\_arg);

update rental set rentalreturneddate = rentalreturneddate\_arg; -- recording that the DVD has been returned

select func\_return\_remainingrentalonetime(memberid\_arg) into rentalchangesnow; -- return the number of additional dvds which can be rented right now

update DVD set dvdquantityonrent = dvdquantityonrent + -1

where dvdid = dvdid\_arg; -- on rental number -1 no matter whether returned or lost because we got this report anyway

if (status\_arg is null) then -- notes the DVD is lost in which case they are charged against their account

insert into payment(memberid, amountpaid, amountpaiddate, amountpaiduntildate)

values (memberid\_arg, paymentamount, rentalreturneddate\_arg, (rentalreturneddate\_arg + interval '1 month' \* 1)); -- I saw amountpaiduntildate always be one month after

update DVD set dvdlostquantity = dvdlostquantity + 1

where dvdid = dvdid\_arg;

else update DVD set dvdquantityonhand = dvdquantityonhand + 1

where dvdid = dvdid\_arg;

end if;

select func\_return\_firstone\_in\_rentalqueue(memberid\_arg) into dvd\_id; -- get a movie from the customer’s request list

call delete\_rentalqueue(memberid\_arg, dvd\_id); -- removes this DVD from the queue

insert into rental(memberid, dvdid, rentalrequestdate) values (memberid\_arg, dvd\_id, rentalreturneddate\_arg); -- create next new rental record once returned

update DVD set dvdquantityonhand = dvdquantityonhand - 1

where dvdid = dvd\_id;

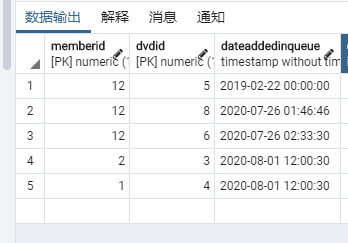
update DVD set dvdquantityonrent = dvdquantityonrent + 1

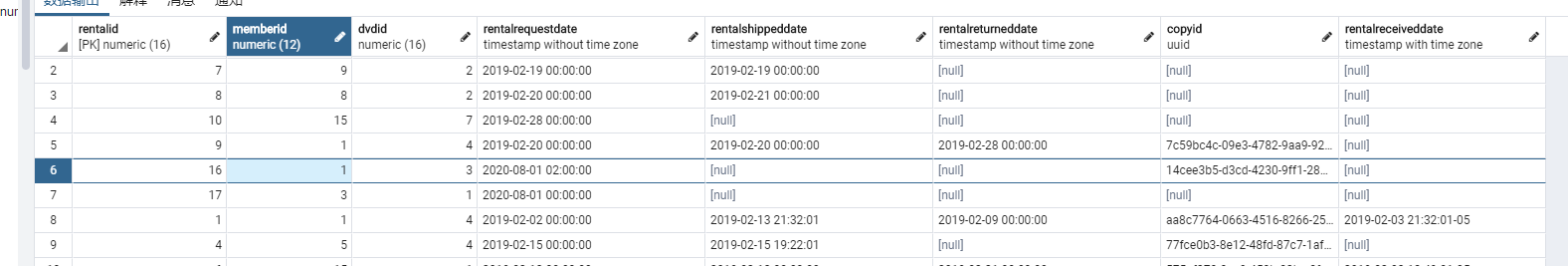
where dvdid = dvd\_id;

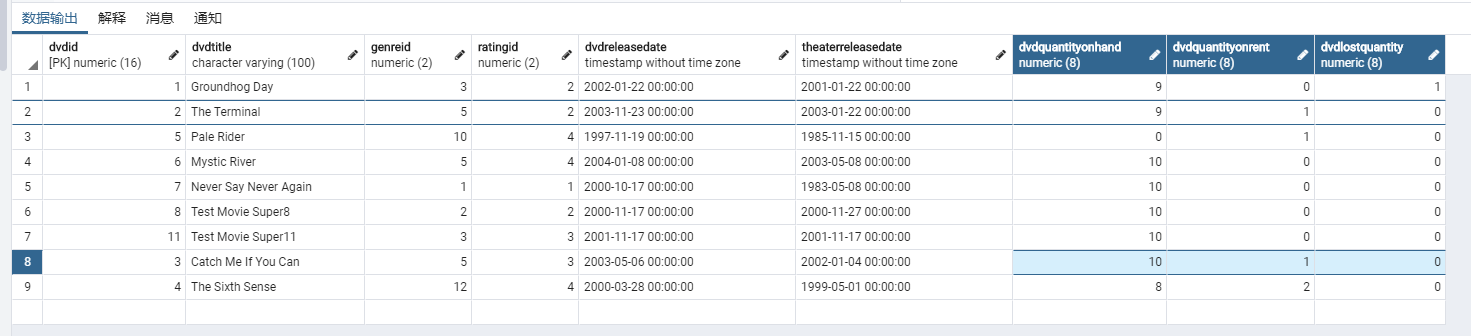
end;

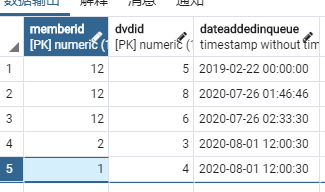
$BODY$;

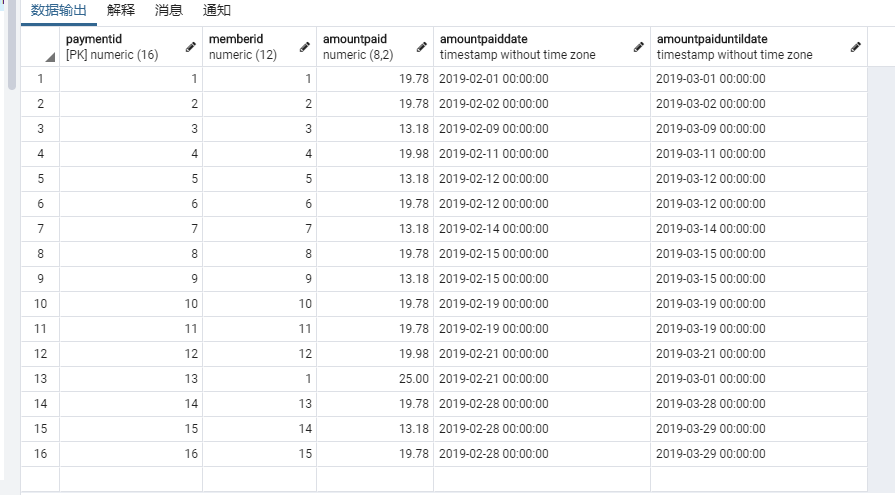
Let’s take a look the situation this moment.









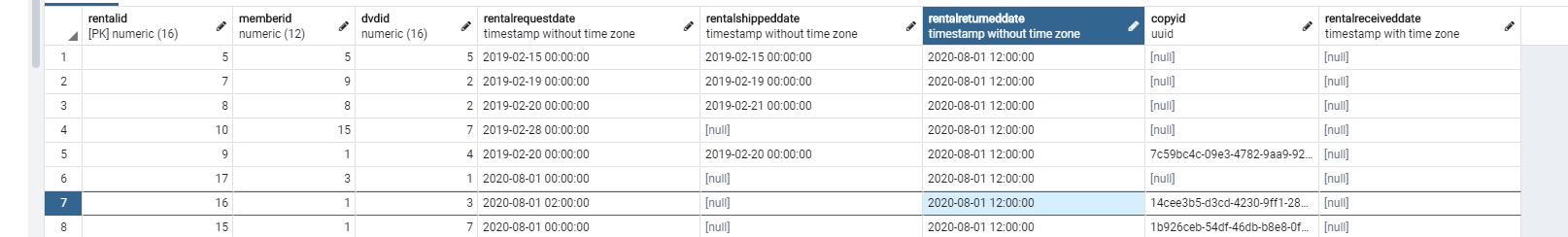


**Let’s begin with this query (customer no.1 is returning rental order 16 with dvd no.3 which lost):**

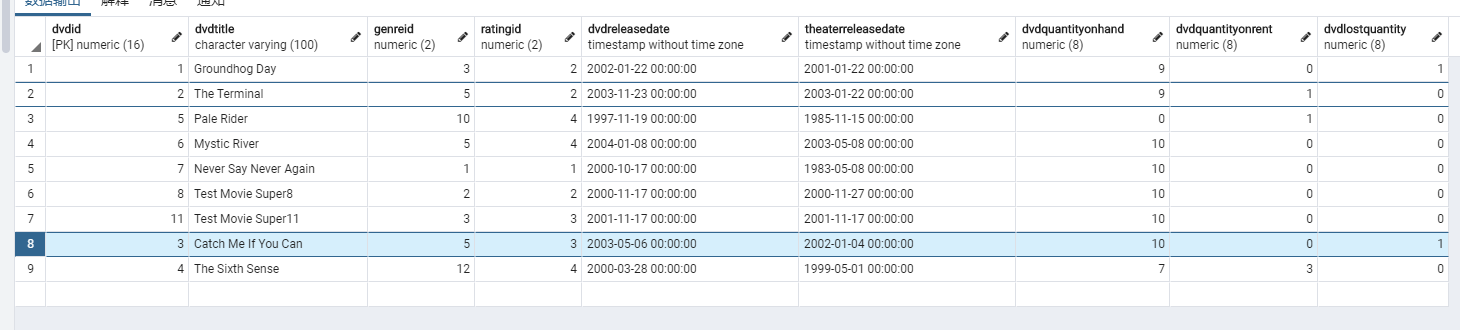
**call rentalqueue\_processing(1,16,3,null,'2020-08-01 12:00:00')**



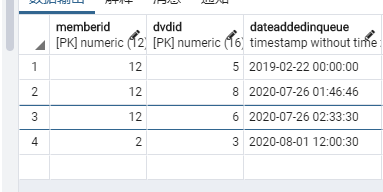
Rental table with return date:



DVD table with DVD quantities:



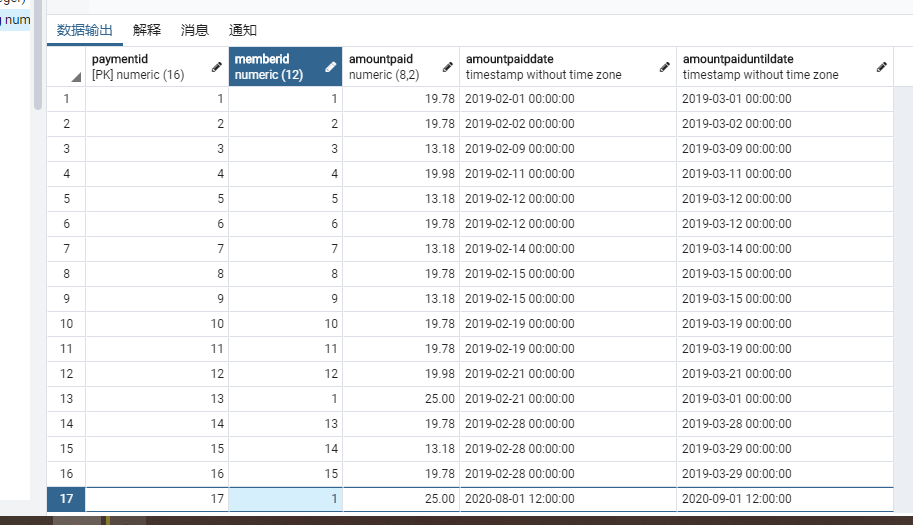
Dequeue in rentalqueue:



Then be added in to rental table:



Add charges by lost:



Everything is automatically applied like magic.

Use the **Ask the Facilitators Discussion Board** if you have any questions regarding the how to approach this assignment.

Save your assignment as ***lastnameFirstname\_assign3.0.docx*** and submit it in the *Assignments* section of the course.

For help uploading files please refer to the *Technical Support* page in the syllabus.