Use this document to capture the screenshots requested in the instructions for assignment 2.1.

**Name**: Mi Gao

**Date: 07/25/2020**

**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. **First, I need to do some extra works for supporting History table. I need to create connections by using trigger to update dvd copy id in rental table; if you feel any confusion about those thing, please check what we did in Assignment 1 for DVD\_COPY.**

CREATE OR REPLACE FUNCTION public.func\_update\_copyid\_in\_rental()

RETURNS trigger

LANGUAGE 'plpgsql'

VOLATILE

COST 100

AS $BODY$begin

begin

UPDATE rental SET copyid = new.copyid --update the copyid on the rental

from dvd\_copy

WHERE new.rentalid = rental.rentalid; --based on the rentalid

return NULL;

end;

$BODY$;

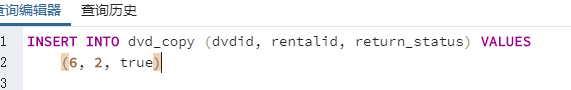
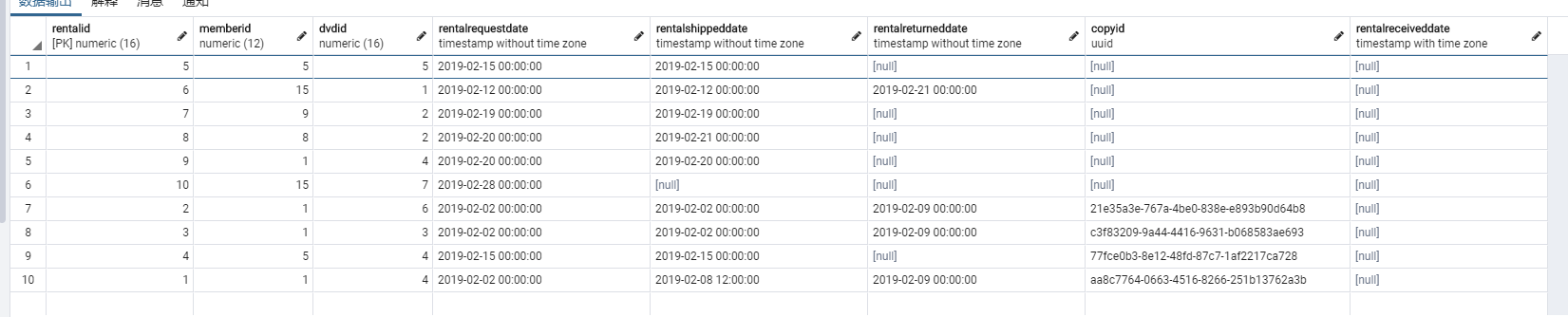
CREATE TRIGGER tri\_update\_copyid\_to\_rental

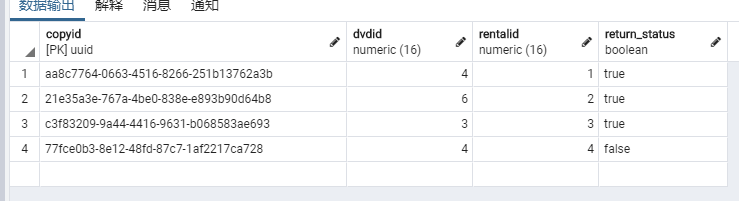
AFTER INSERT OR DELETE OR UPDATE

ON public.dvd\_copy

FOR EACH ROW

EXECUTE PROCEDURE public.func\_update\_copyid\_in\_rental();



1. **Create a History table.**

When I was planning it, I designed it as the denormalization combined some other information rather than things in the rental at first time.

**CREATE TABLE public."history"**

**(**

**historyid numeric(16) NOT NULL,**

**dvdid numeric(16) NOT NULL,**

**memberid numeric(12) NOT NULL,**

**copyid uuid NOT NULL,**

**memberfirstname character varying(32) NOT NULL,**

**memberlastname character varying(32) NOT NULL,**

**"RentalShippedDate" timestamp(3) without time zone,**

**"RentalreceivedDate" timestamp(3) without time zone,**

**"RentalReturnedDate" timestamp(3) without time zone,**

**CONSTRAINT "History\_HistoryId\_PK" PRIMARY KEY (historyid),**

**CONSTRAINT "History\_DVDId\_FK" FOREIGN KEY (DVDId) REFERENCES DVD(DVDId),**

**CONSTRAINT "History\_memberid\_FK" FOREIGN KEY (memberid) REFERENCES member(memberid),**

**CONSTRAINT "History\_copyid\_FK" FOREIGN KEY (copyid) REFERENCES member(copyid),**

**)**

**WITH (**

**OIDS = FALSE**

**);**

**ALTER TABLE public."history"**

**OWNER to postgres;**

However, I remade it. In my opinion, this should be an **audit** **table** at first and that’s why we need it. Which mean, we are going to oversee every actions by using this table. Otherwise, it will be just an extension version of the rental table; it’s not interesting. Thus, I made this.

**CREATE TABLE history**

**(**

**historyid numeric(16) NOT NULL,**

**dvdid numeric(16) NOT NULL,**

**memberid numeric(12) NOT NULL,**

**rentalid numeric(16) NOT NULL,**

**copyid uuid NOT NULL,**

**"time" timestamp(3) without time zone NOT NULL,**

**"action" character varying(20) NOT NULL,**

**CONSTRAINT "History\_HistoryId\_PK" PRIMARY KEY (historyid),**

**CONSTRAINT "History\_DVDId\_FK" FOREIGN KEY (DVDId) REFERENCES DVD(DVDId),**

**CONSTRAINT "History\_memberid\_FK" FOREIGN KEY (memberid) REFERENCES member(memberid),**

**CONSTRAINT "History\_rentalid\_FK" FOREIGN KEY (rentalid) REFERENCES rental(rentalid),**

**CONSTRAINT "History\_copyid\_FK" FOREIGN KEY (copyid) REFERENCES dvd\_copy(copyid)**

**)**

**WITH (**

**OIDS = FALSE**

**);**

**ALTER TABLE public."history"**

**OWNER to postgres;**

|  |  |
| --- | --- |
| Historyid | Primary key for this table |
| Dvdid | Foreign key referred to the dvd table |
| Memberid | Foreign key referred to the member table |
| rentalid | Foreign key referred to the rental table |
| Copyid | Foreign key referred to the dvd copy table |
| Memberfirstname | Binned with members |
| Memberlastname | Binned with members |
| Time | Time stamp, depends on which column has been updated |
| Action | Action records such as “shipped” or “returned” which depend on which column have been updated |

Of cause, I will add rentalreceiveddate attribute to the rental. Then, I will make time and action attributes to be working by using triggers in next few questions.

Anyway, I need to give it a sequence combined with to generate historyid first.

CREATE SEQUENCE historyid\_sequence

start 1

increment 1;

ALTER TABLE history ALTER COLUMN historyid SET DEFAULT nextval('historyid\_sequence')

1. **Implement a trigger for this new rental history table that prevents deletions from the table using error handling logic.**

Actually, in my opinion, this question is meaning that I need to create a trigger to tell whoever try to delete tuples from our audit table that is not ok so that we can protect our records. I hope I did not understand that wrong. Also, I did question 3 and 4 before this question.

CREATE OR REPLACE FUNCTION public.func\_deletion\_error\_handling\_history()

RETURNS trigger

LANGUAGE 'plpgsql'

VOLATILE

COST 100

AS $BODY$

begin

if TG\_OP='DELETE' then --check the condition of “delete”

raise EXCEPTION 'Error, History is forbidden to be deleted.'; --alert the exception

return old;

end if;

end;

$BODY$;

CREATE TRIGGER tri\_deletion\_error\_handling

BEFORE DELETE --alert before deletion actived

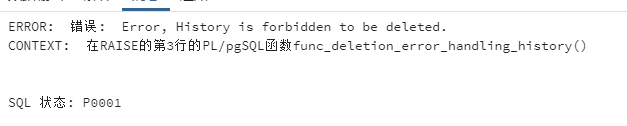
ON public.history

FOR EACH ROW

EXECUTE PROCEDURE public.func\_deletion\_error\_handling\_history();

**Then, let’s try:**

delete from history where historyid = 20





**Looks everything is ok.**

1. **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.**

CREATE OR REPLACE FUNCTION public.func\_update\_rentalshippeddate()

RETURNS trigger

LANGUAGE 'plpgsql'

VOLATILE

COST 100

AS $BODY$

begin

Delete from history where rentalid = new.rentalid and action = 'shipped'; --remove duplications

INSERT INTO history(dvdid, memberid, rentalid, copyid, time, action) --because I set the sequence for the table so that I need those inputs only

select rental.dvdid, rental.memberid, rental.rentalid,

rental.copyid, new.rentalshippeddate,

(CASE WHEN rentalshippeddate is NULL THEN 'unshipped'

Else 'shipped' END) from rental --Used CASE WHEN to see values for our actions; unshipped type may not needed but I just created for exercising this part knowledge

where new.rentalid = rental.rentalid; --based on rentalid to bin

return NULL;

end;

$BODY$;

CREATE TRIGGER tri\_update\_shipped\_in\_history

BEFORE UPDATE OF rentalshippeddate --only checking the update on this column

ON public.rental

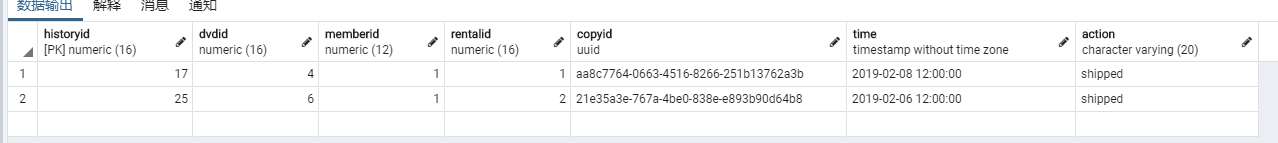
FOR EACH ROW

EXECUTE PROCEDURE public.func\_update\_rentalshippeddate();

**Try this:**

update rental set rentalshippeddate = '2019-02-06 12:00:00' where rentalid = 2





**Looks good. And by the way, the sequence on output table is 1 and 2 but id already been 17 and 25 is because I set the auto sequence for history table first and test many times of error. Also, this trigger have good functionality without deletion error handler trigger but will get expression when delection trigger on because I wrote the delete query for deduplication at the first:**

Delete from history where rentalid = new.rentalid and action = 'shipped';

**However, if we do not care about the update of shipped date fixing, we can just remove that line of code and keep every line of record in our audit table as well. Thus, I just leave that line of query here so far for let you know how I think about this question could be in my brain.**

**Second edition:**

I made update for the deduplication after I read more documents about the things work like event listeners. Now, we can use deletion error handing trigger and do not change on the trigger then work with this new function together!

begin

if (TG\_OP = 'INSERT') then

INSERT INTO history(dvdid, memberid, rentalid, copyid, time, action) --because I set the sequence for the table so that I need those inputs only

select rental.dvdid, rental.memberid, rental.rentalid,

rental.copyid, new.rentalshippeddate,

(CASE WHEN rentalshippeddate is NULL THEN 'unshipped'

Else 'shipped' END) from rental --Used CASE WHEN to see values for our actions; unshipped type may not needed but I just created for exercising this part knowledge

where new.rentalid = rental.rentalid; --based on rentalid to bin

end if;

if (TG\_OP = 'UPDATE') then

if not EXISTS (select 1 from history

where history.rentalid = new.rentalid and action = 'shipped') then

INSERT INTO history(dvdid, memberid, rentalid, copyid, time, action) --because I set the sequence for the table so that I need those inputs only

select rental.dvdid, rental.memberid, rental.rentalid, rental.copyid, new.rentalshippeddate,

(CASE WHEN rentalshippeddate is NULL THEN 'unshipped'

Else 'shipped' END) from rental --Used CASE WHEN to see values for our actions; unshipped type may not needed but I just created for exercising this part knowledge

where new.rentalid = rental.rentalid; --based on rentalid to bin

end if;

if OLD.rentalshippeddate <> NEW.rentalshippeddate then -- if updated time different with old one

update history set time = new.rentalshippeddate from rental

where history.rentalid = new.rentalid and action = 'shipped';

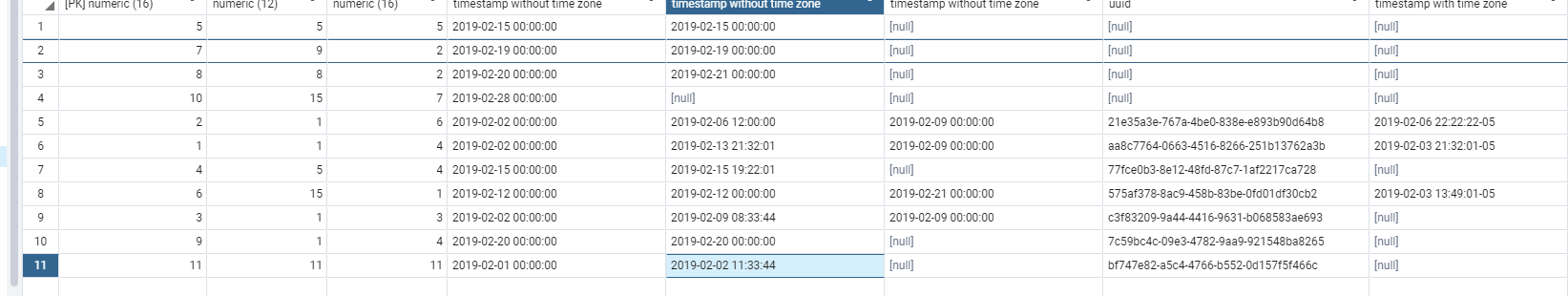
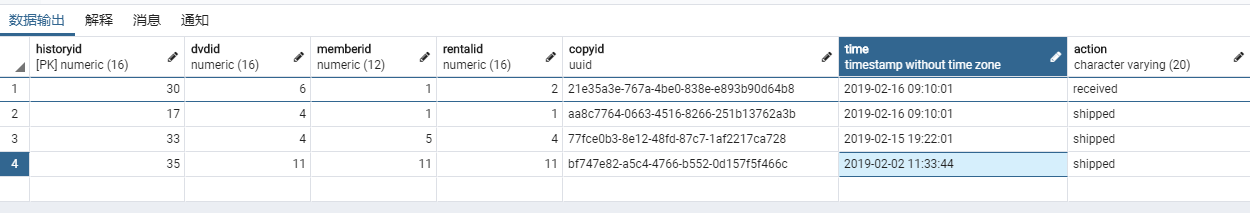
end if;

end if;

return new;

end;



The reason I set the fire time at the update rather than whole tuple inserted is based on the logic of the actual operation; I mean operation should be based on the order from request to the shipment then received, and then the return. Also this is why only the request date set as not null value I think. And if the situation is one whole completed records row with has been inserted in to the rental table, I think it will not be necessary to be recorded on our audit table as well.

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

CREATE OR REPLACE FUNCTION public.func\_update\_rentalreceiveddate()

RETURNS trigger

LANGUAGE 'plpgsql'

VOLATILE

COST 100

AS $BODY$begin

INSERT INTO history(dvdid, memberid, rentalid, copyid, time, action) --because I set the sequence for the table so that I need those inputs only

select rental.dvdid, rental.memberid, rental.rentalid,

rental.copyid, new.rentalreceiveddate,

(CASE WHEN rentalreceiveddate is NULL THEN 'unreceived'

Else 'received' END) from rental --Used CASE WHEN to see values for our actions; unreceived type may not needed but I just created for exercising this part knowledge

where new.rentalid = rental.rentalid; --based on rentalid to bin

return NULL;

end;

$BODY$;

CREATE TRIGGER tri\_update\_received\_in\_history

Before UPDATE OF rentalreceiveddate --only checking this column

ON public.rental

FOR EACH ROW

EXECUTE PROCEDURE public.func\_update\_rentalreceiveddate();

**Then try this:**

update rental set rentalreceiveddate = '2019-02-07 12:00:00' where rentalid = 2

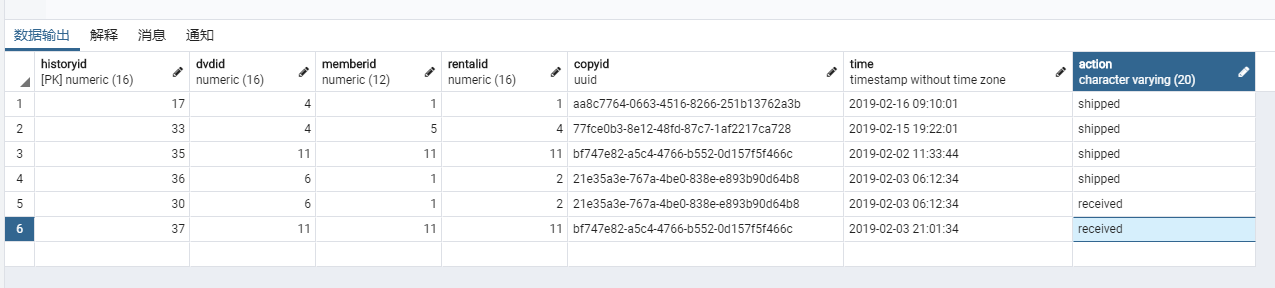




Good. Almost same with how shipping trigger works. I just removed deduplication query this time, but you can still add it if you really need and want. As long as we are building an audit table, I think I will record any special detail happened then check this table whenever I need some clues. For example, why that duplicate information exist and from who.

**Second edition:**

I also made update for the deduplication. Now, we can use deletion error handing trigger with this new function together also!

begin

if (TG\_OP = 'INSERT') then

INSERT INTO history(dvdid, memberid, rentalid, copyid, time, action) --because I set the sequence for the table so that I need those inputs only

select rental.dvdid, rental.memberid, rental.rentalid,

rental.copyid, new.rentalreceiveddate,

(CASE WHEN rentalreceiveddate is NULL THEN 'unreceived'

Else 'received' END) from rental --Used CASE WHEN to see values for our actions; unreceived type may not needed but I just created for exercising this part knowledge

where new.rentalid = rental.rentalid; --based on rentalid to bin

end if;

if (TG\_OP = 'UPDATE') then

if not EXISTS (select 1 from history

where history.rentalid = new.rentalid and action = 'received') then

INSERT INTO history(dvdid, memberid, rentalid, copyid, time, action) --because I set the sequence for the table so that I need those inputs only

select rental.dvdid, rental.memberid, rental.rentalid, rental.copyid, new.rentalreceiveddate,

(CASE WHEN rentalreceiveddate is NULL THEN 'unreceived'

Else 'received' END) from rental --Used CASE WHEN to see values for our actions; received type may not needed but I just created for exercising this part knowledge

where new.rentalid = rental.rentalid; --based on rentalid to bin

end if;

if OLD.rentalreceiveddate <> NEW.rentalreceiveddate then -- if updated time different with old one

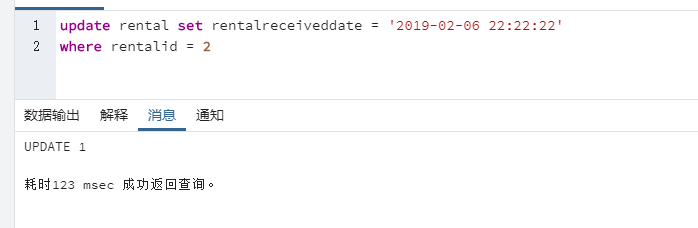
update history set time = new.rentalreceiveddate from rental

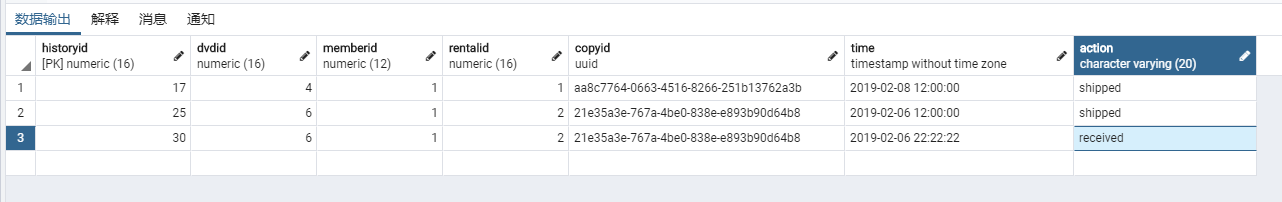
where history.rentalid = new.rentalid and action = 'received';

end if;

end if;

return new;

end;



1. 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. You will need to add some error handling in your code.

I made it to be two parts; the first one called add\_rentalqueue which used for add into the list, and other one called priority\_rentalqueue which used for increase the priority of the movie. For this, I added the queue\_num attribute on rentalqueue table first.

CREATE OR REPLACE PROCEDURE public.add\_rentalqueue(

IN memberid\_arg numeric, -- memberid add in the list with that dvdid

IN dvdid\_arg numeric, -- new dvdid added by user in the queue

IN dateaddedinqueue\_arg timestamp without time zone DEFAULT current\_timestamp(0), -- the date when that DVD be added into the queue, and I set this as current\_timestamp(0) because the data I got actually are all timestamp(0) although they set as (3)

IN queue\_arg integer DEFAULT 0) -- set order or not

LANGUAGE 'plpgsql'

AS $BODY$begin

if EXISTS (select 1 from rentalqueue where memberid = memberid\_arg and dvdid = dvdid\_arg) then -- the condition of existing or not

raise EXCEPTION 'Error, can not add same movie twice at the list.';

else insert into rentalqueue (memberid, dvdid, dateaddedinqueue, queue\_num)

values(memberid\_arg, dvdid\_arg, dateaddedinqueue\_arg, queue\_arg); -- add into the rentalqueue

end if;

end;

$BODY$;

CREATE OR REPLACE PROCEDURE priority\_rentalqueue(

IN memberid\_arg numeric, -- memberid add in the list with that dvdid

IN dvdid\_arg numeric, -- new dvdid added by user in the queue

IN queue\_arg integer DEFAULT 1) -- set it to be 1 based on the default was 1 when they have been inserted

LANGUAGE 'plpgsql'

AS $BODY$

begin

if EXISTS (select 1 from rentalqueue where memberid = memberid\_arg and dvdid = dvdid\_arg) then -- the condition of existing or not

if (SELECT queue\_num from rentalqueue

where memberid = memberid\_arg and dvdid = dvdid\_arg) = 0 --value check

then

update rentalqueue set queue\_num = queue\_arg --default value

where memberid = memberid\_arg and dvdid = dvdid\_arg;

else update rentalqueue set queue\_num = queue\_num + 1 --otherwise, if it was not 0, just add 1 level up

where memberid = memberid\_arg and dvdid = dvdid\_arg;

end if;

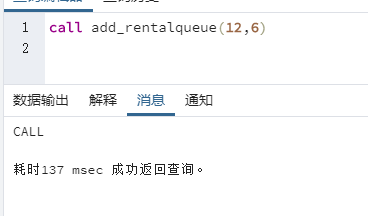
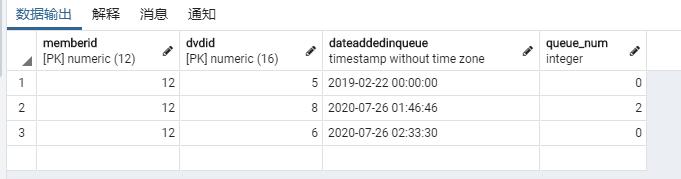
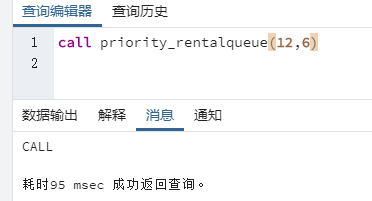
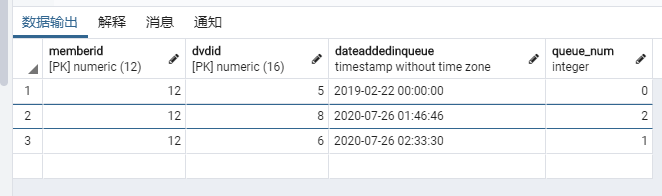
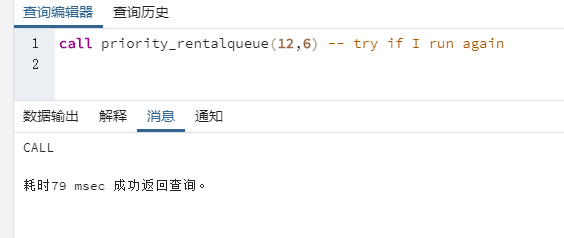
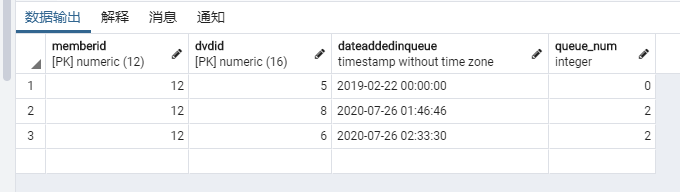
else raise EXCEPTION 'Error, can not find the movie at the list.';

end if;

end;

$BODY$;

The reason I do like this is based on two point: time and priority. As we knew, we have a default time set when we first time insert so that we can use ORDER BY to generate the order of the list of the customer. Then, we double check with my rentalqueue column which was new added which one has the biggest priority number.

This is more suitable for multiple clicks on buttons of using GUI interface. There is an update version which more significant to add priority:

begin

if EXISTS (select 1 from rentalqueue where memberid = memberid\_arg and dvdid = dvdid\_arg) then

if (SELECT queue\_num from rentalqueue

where memberid = memberid\_arg and dvdid = dvdid\_arg) = 0

then

update rentalqueue set queue\_num = queue\_arg

where memberid = memberid\_arg and dvdid = dvdid\_arg;

elseif (queue\_arg > 1) then update rentalqueue set queue\_num = queue\_num + queue\_arg

where memberid = memberid\_arg and dvdid = dvdid\_arg;

else update rentalqueue set queue\_num = queue\_num + 1

where memberid = memberid\_arg and dvdid = dvdid\_arg;

end if;

else raise EXCEPTION 'Error, can not find the movie at the list.';

end if;

end;

However, this is not my original plan. My original plan which had been abandoned was to create an automatic increment sequence for entire table and each tuple no matter who. Then, if someone want to modify the priority of one movie, I will use MIN() -1 to generate the new minimum value of queue\_num under his ID. But for some reason, Aggregate function can not be used in values() and other Multi-level nesting I created was irritating me so that I gave up that plan. Wish you can give me some advices to implement the way like using MAX() to quickly apply the extremum for reducing steps.

1. 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.

This one was much easier than above if I made it right.

CREATE OR REPLACE PROCEDURE delete\_rentalqueue(

IN memberid\_arg numeric, -- memberid add in the list with that dvdid

IN dvdid\_arg numeric) -- new dvdid added by user in the queue

LANGUAGE 'plpgsql'

AS $BODY$

begin

if EXISTS (select 1 from rentalqueue where memberid = memberid\_arg and dvdid = dvdid\_arg) then

delete from rentalqueue where memberid = memberid\_arg and dvdid = dvdid\_arg;

else raise EXCEPTION 'Error, can not find the movie at the list.';

end if;

end;

$BODY$;

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

Save your assignment as ***lastnameFirstname\_assign2.1.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.