Use this document to capture the screenshots requested in the instructions for assignment 1.1.

**Name**: Mi Gao

**Date: 07/18/2020**

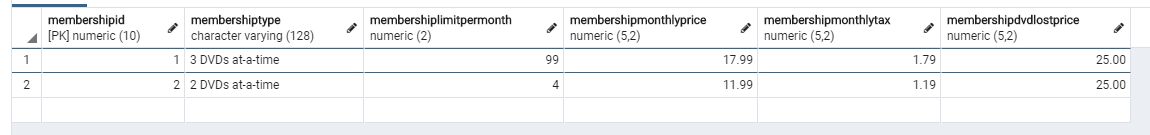
**The submission template uses the following format :**

* 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.

Part 1: The Netflix Schema, DML and DDL

1. [ Learning objectives: adding and populating columns, SQL string operators]  
   Examine the Membership table. Note that membership types include information about how many DVDs a member can rent at a time, and that there is also a limit of rentals per month.
2. Select all data from the Membership table and review it.

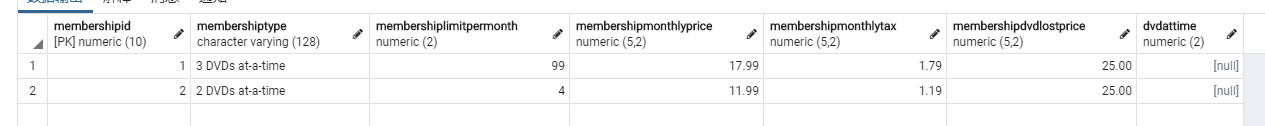
**SELECT \* FROM membership**



1. Add an attribute called DVDAtTime that represents the number of DVDs that a member can rent at one time. For example, the record with the name ‘3 DVDs at-a-time’ should have a value of ‘3’ in the DVDAtTime column.

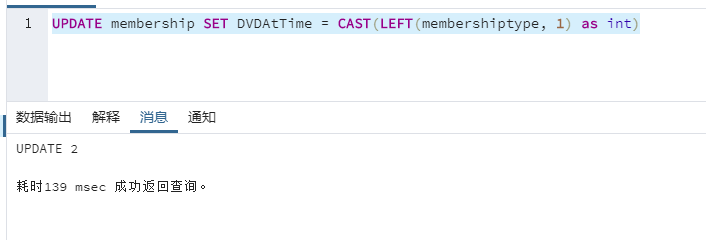
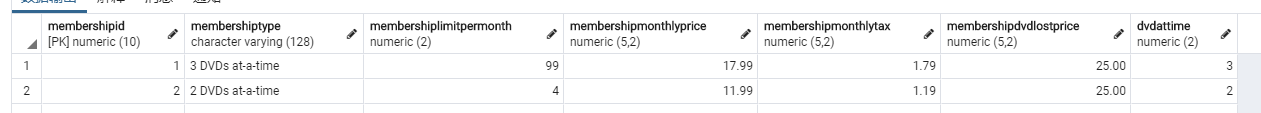
**ALTER TABLE membership**

**ADD COLUMN DVDAtTime numeric(2)**



1. The names in the Membership table data provided all begin with a digit representing the number of DVDs that a member can have out at a time. Use this to update the DVDAtTime attribute to be consistent with the MembershipType using a built-in substring function (left()); also using cast() to transfer the data type (string) returned from left() to numeric (integer).

**UPDATE membership SET DVDAtTime = CAST(LEFT(membershiptype, 1) as int)**

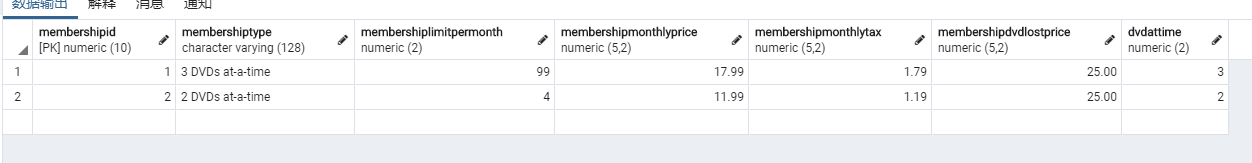
 

**(PS 1: I just want to state that this part can be done better with the trigger for implementing automatically once anything updated on “membershiptype” in the future. I did not do those because I need to save time on other questions.)**

**(PS 2: Sorry for my pgAdmin4 language showing; it was matched with my system default language. However, you will see the output format returned will be all same with other languages. If you feel any trouble with reading those output, please let me know then I will fix them.)**

1. Select all data from the membership table to verify results.

**SELECT \* FROM public.membership**

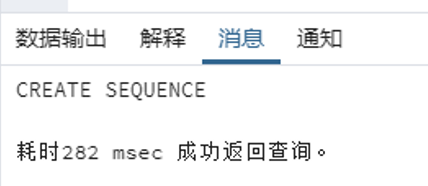


1. [ Learning objectives: sequences]
2. Implement a single sequence on one of the primary key for the rental table.

**CREATE SEQUENCE rentalid\_sequence**

**start 11**

**increment 1;**



1. You should not need to delete data from the tables or change any of the current synthetic primary keys. There is a parameter for the initial starting value of a sequence that you will need to set, based on the legacy synthetic keys, when defining each sequence.

**Based on this request, I do not change anything I have from the original data then set my new sequence start from 11 which continue to count after them.**

1. Demonstrate sequence by inserting two new records into the rental table.

**INSERT INTO rental**

**(rentalid, memberid, dvdid, rentalrequestdate, rentalshippeddate, rentalreturneddate)**

**VALUES**

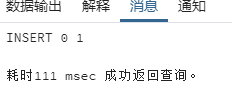
**(nextval('rentalid\_sequence'), 2, 2, '2019-02-02 00:00:00', '2019-02-02 00:00:00', '2019-02-09 00:00:00');**

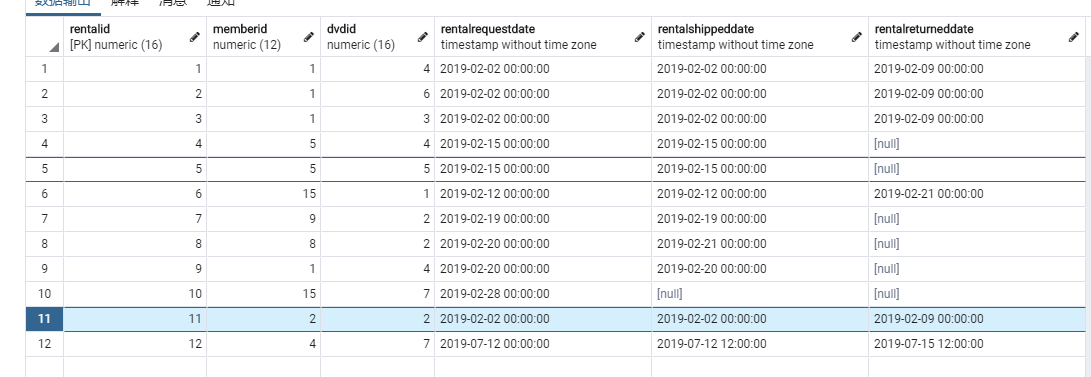
**INSERT INTO rental**

**(rentalid, memberid, dvdid, rentalrequestdate, rentalshippeddate, rentalreturneddate)**

**VALUES**

**(nextval('rentalid\_sequence'), 4, 7, '2019-07-12 00:00:00', '2019-07-12 12:00:00', '2019-07-15 12:00:00');**





Also, I tried to use second way:

**ALTER TABLE rental ALTER COLUMN rentalid SET DEFAULT nextval('rentalid\_sequence');**

This will set rentalid attribute be automatically using my sequence.

1. [ Learning objectives: schema augmentation, nullability, CHECK]
2. Create foreign key constraints on DVDREVIEW to the DVD and the MEMBER tables to assure that the members and DVDs are valid.

CONSTRAINT DVDREVIEW\_MemberId\_FK FOREIGN KEY (MemberId) REFERENCES Member(MemberId),

CONSTRAINT DVDREVIEW\_DVDId\_FK FOREIGN KEY (DVDId) REFERENCES DVD(DVDId));

1. DVDREVIEW is to have an integer attribute called STARVALUE, which represents the number of stars a member assigns to a DVD; this value cannot be null.

STARVALUE int NOT NULL,

1. DVDREVIEW will have a REVIEWDATE of the review, which cannot be null and defaults to today’s date.

REVIEWDATE TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,

**(I used TIMESTAMP(3) because I saw all setting for time are using this timestamp which stores milliseconds. )**

1. DVDREVIEW will also have an optional comment attribute so that a member can leave a written review.

COMMENTVALUE text,

1. DVDREVIEW will have a check constraint on the STARVALUE attribute that only allows values from zero to five.

STARVALUE int NOT NULL CHECK (STARVALUE>=0) CHECK (STARVALUE<=5),

1. Use a sequence to manage primary key.

At the beginning, I planned to do like this for primary key:

DvdReviewId numeric(10) NOT NULL DEFAULT nextval('dvdreviewid\_sequence'),

However, I fixed my mine once I re-read the role of this table because I think it will be useless to have an individual key or id for each one based on the request of each customer can rate for a movie once. I decided to create composite key which depends on DVDId and MemberId as primary key so that this will also figure out the request. Instead of that, I will make sequence for both DVD and Member table just like what I did for the rental table; for example, **ALTER TABLE dvd ALTER COLUMN dvdid SET DEFAULT nextval('dvdid\_sequence').**

Therefore:

**CREATE TABLE DVDREVIEW(**

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

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

**STARVALUE int NOT NULL CHECK (STARVALUE>=0) CHECK (STARVALUE<=5),**

**REVIEWDATE TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,**

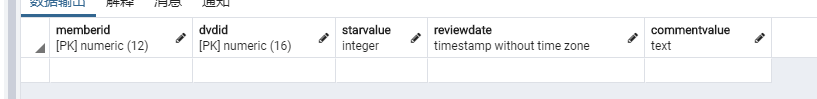
**COMMENTVALUE text,**

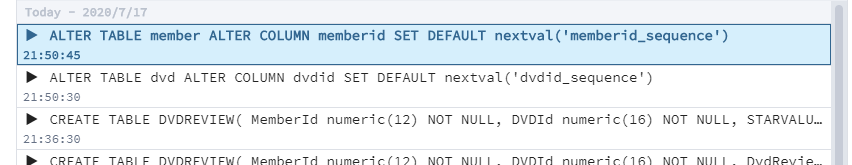
**CONSTRAINT DVDREVIEW\_MemberId\_DVDId\_PK PRIMARY KEY (MemberId,DVDId),**

**CONSTRAINT DVDREVIEW\_MemberId\_FK FOREIGN KEY (MemberId) REFERENCES Member(MemberId),**

**CONSTRAINT DVDREVIEW\_DVDId\_FK FOREIGN KEY (DVDId) REFERENCES DVD(DVDId));**

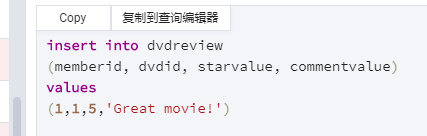


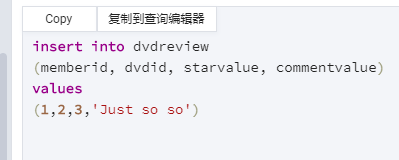




1. [ Learning objectives: testing schema augmentation, DBMS date function, view, code reuse]
2. Insert three records into the DVDREVIEW for members and a movie of your choice.

Those two will get date and time by what we already set when create the table above.





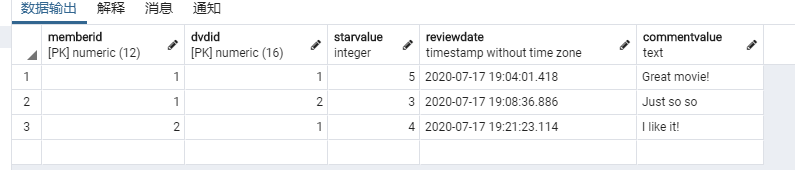
This used DBMS function now():

**insert into dvdreview**

**(memberid, dvdid, starvalue, reviewdate, commentvalue)**

**values**

**(2,1,4,now(),'I like it!')**



1. Write a view that returns the following columns: a concatenated Member Name containing the Member’s First and Last Name, the Title of the DVD, and the Member’s Review including STARVALUE, REVIEWDATE and Comment. Avoid using old-style joins where the join constraints are in WHERE clauses, because these are not compliant with current good practice or ANSI standards. Instead, use the JOIN keyword. Demonstrate that the view works by selecting a review of your choice for a particular movie.

CREATE VIEW review\_view as

SELECT member.memberfirstname, member.memberlastname, dvd.dvdtitle, dvdreview.starvalue, dvdreview.reviewdate, dvdreview.commentvalue

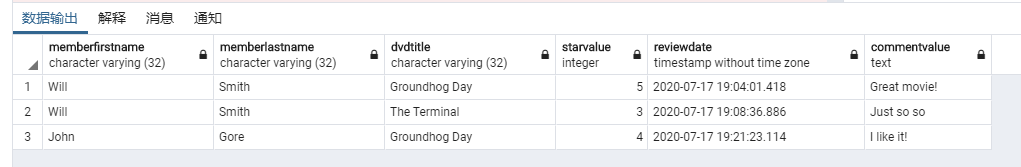
FROM dvdreview

INNER JOIN dvd ON

dvdreview.dvdid = dvd.dvdid

INNER JOIN member ON

dvdreview.memberid = member.memberid



SELECT

\*

FROM

review\_view

WHERE

commentvalue = 'I like it!'



1. Write an insert statement that tries to insert a review that violates the STARVALUE check constraint.

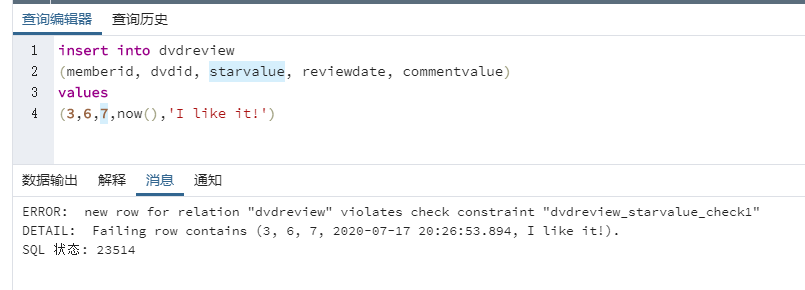
**Error will be thrown; test case: starvalue = 7.**

insert into dvdreview

(memberid, dvdid, starvalue, reviewdate, commentvalue)

values

(3,6,7,now(),'I like it!')



1. Suppose that a member of your choice changed their mind about a review for a specific DVD. Write an update statement to change a specific customer’s review for a specific DVD of your choice. Run the view above (b) to verify your results.

**View also will be updated with how the source of the table goes.**

**Test case: Will Smith (memberid 1) and *The Terminal* (dvdid 2)**

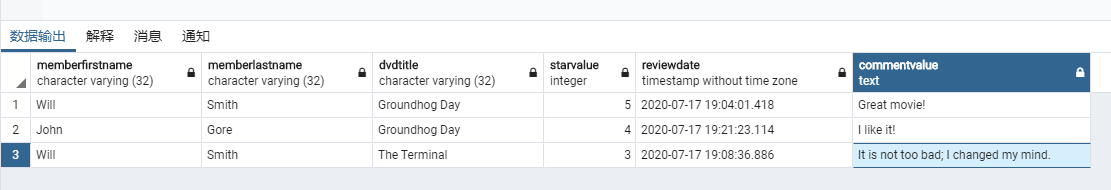
update dvdreview

set

commentvalue = 'It is not too bad; I changed my mind.'

where

memberid = '1' and dvdid = '2'



1. Suppose that a member changed their mind about another review for a specific DVD and they want it deleted. Write a delete statement to delete a specific customer’s review for a specific DVD of your choice. Run view above (b) to verify your results.

**View also will be updated with how the source of the table goes.**

**Test case: Will Smith (memberid 1) and *The Terminal* (dvdid 2)**

DELETE FROM dvdreview WHERE memberid = '1' and dvdid = '2';



1. [ Learning objectives:schema extension]  
   Please note that once you make changes to your schema, you will need to solve the subsequent problems with the added changes. The DVD rental business is having difficulty keeping track of the physical DVDs. They are asking you to extend the schema to track individual physical DVDs. Implement a table called DVD\_COPY. You have to work with the existing data in the Rental and DVD tables and make sure the changes take into account existing DVD quantities. Your solution will need to show updates to the records in the Rental table in order to accommodate the changed referential integrity.
   1. Implement the DVD Copy table.

First:

create extension "uuid-ossp";

Then:

CREATE TABLE DVD\_COPY(

CopyId uuid not null default uuid\_generate\_v4(),

DVDId numeric(16) NOT NULL,

rentalid numeric(16),

return\_status boolean NOT NULL,

CONSTRAINT DVD\_COPY\_CopyId\_PK PRIMARY KEY (CopyId),

CONSTRAINT DVD\_COPY\_DVDId\_FK FOREIGN KEY (DVDId) REFERENCES DVD(DVDId),

CONSTRAINT DVD\_COPY\_rentalid\_FK FOREIGN KEY (rentalid) REFERENCES rental(rentalid));

|  |  |
| --- | --- |
| CopyId | Default Unique ID (uuid) for tracking each hard copy of DVD which has the history for the rental to the customer. |
| DVDId | Combined with the movie in DVD table |
| rentalid | Combined with the rental |
| return\_status | Boolean type, check the status of this copy whether returned or not so that it can help to figure out what happen for this copy and how to do for next step of that rental record. |



Then, I am not fully understand the mean of “solution will need to show updates to the records in the Rental table in order to accommodate the changed referential integrity”; however, I do this for give the same value from DVD COPY table:

alter table rental add column CopyId uuid

UPDATE rental

SET copyid = dvd\_copy.copyid from dvd\_copy

WHERE rental.rentalid = dvd\_copy.rentalid;

ALTER TABLE rental ADD CONSTRAINT rental\_copyid\_fk FOREIGN KEY (copyid) REFERENCES dvd\_copy (copyid) MATCH FULL;

* 1. Populate the DVD copy table with a reasonable data set – it does not need to match exactly the number of DVDs in the DVD table – but needs to make the Rental table consistent with the copies you added.

According the request that “it does not need to match exactly the number of DVDs in the DVD table,” so that I made this on DVD copies based on what recorded in rental table.

INSERT INTO dvd\_copy (dvdid, rentalid, return\_status) VALUES

(4, 1, true),

(6, 2, true),

(3, 3, true),

(4, 4, false),

(5, 5, false),

(1, 6, true),

(2, 7, false),

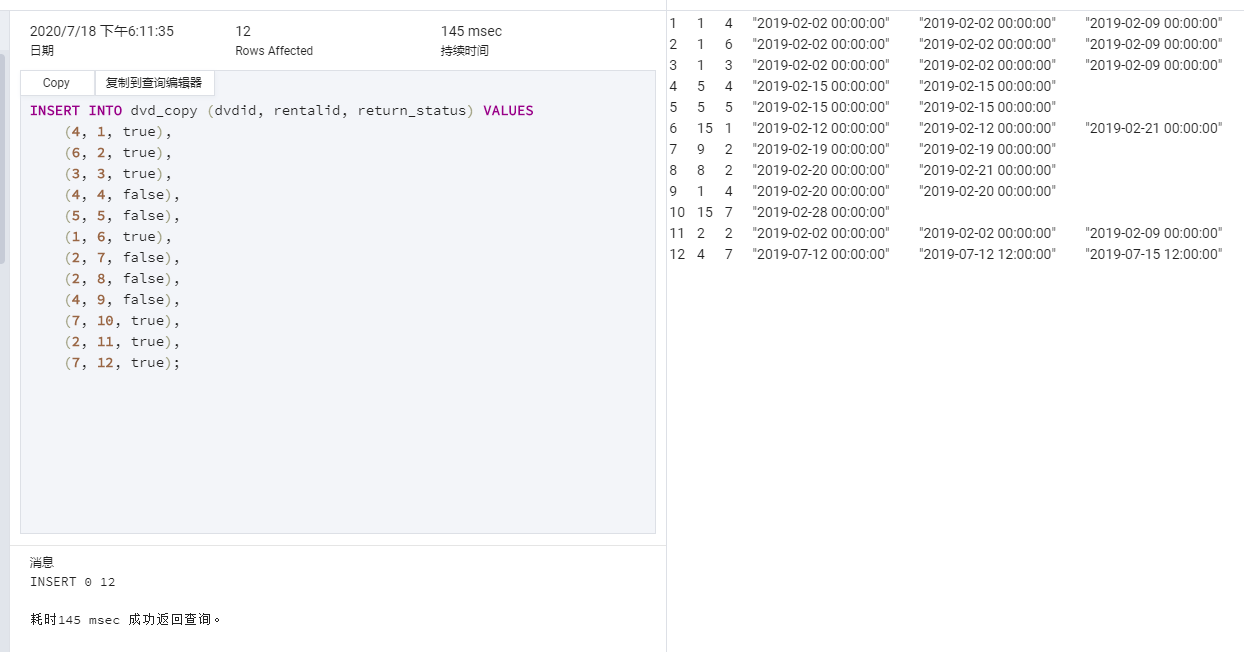
(2, 8, false),

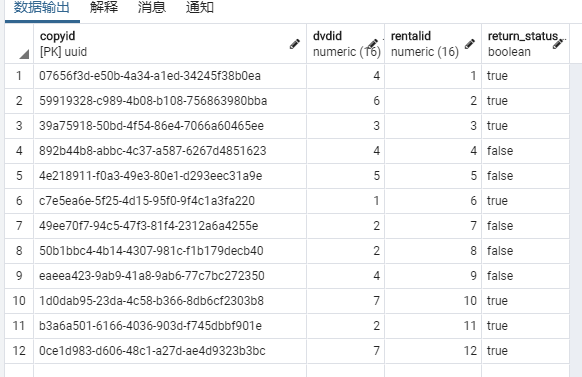
(4, 9, false),

(7, 10, true),

(2, 11, true),

(7, 12, true);





* 1. Explain why you left the DVD attribute in the DVD table, or why you removed it, and what would the pros and cons be of your approach.

**For the normalization, otherwise it will make data redundancy and repeat. For example, if I removed DVD attribute in DVD table, there are going to show bunch of same name for each kind of copy of each DVD. Also, I thought that I do not need to make tuples for each copy yet; instead, I made for each copy we have the record for the rental which means I only need to take a watch on what we rented to customer at this moment. However, I thought it also is fine for insert whole copies’ record we have in the stock if we want.**

# Part 2: Joins and Subqueries

1. [ Learning objectives: review of SELECT with joins]  
   Write a select statement that lists the Title, Genre, Rating, MoviePersons and their Roles of DVDs for a Genre of your choice. The results should all be natural strings, and not integer IDs.

**select dvd.dvdtitle, genre.genrename, rating.ratingname, movieperson.personfirstname, movieperson.personlastname, role.rolename**

**from dvd**

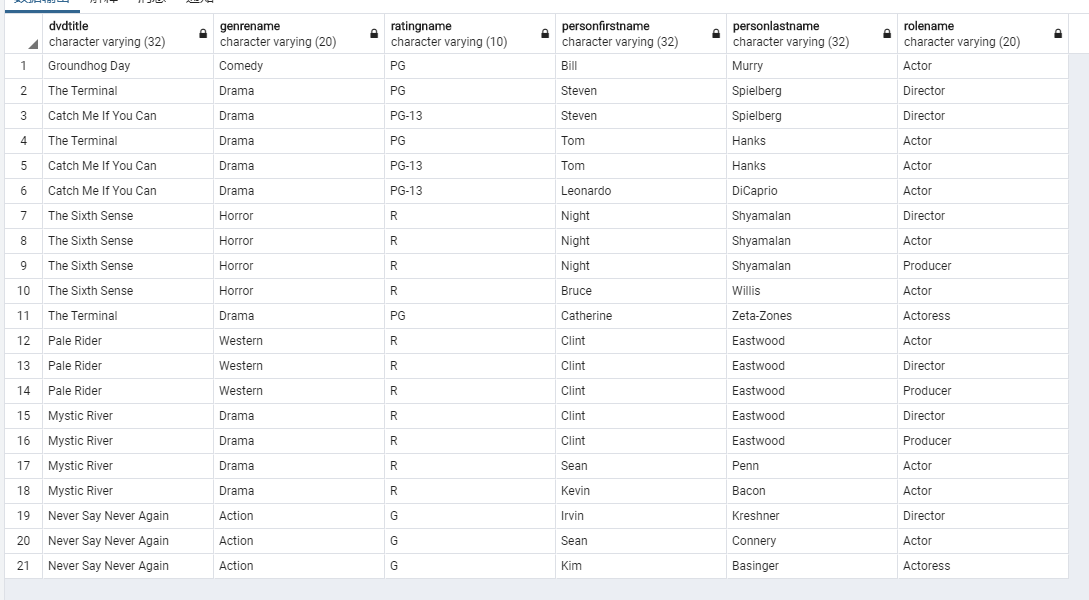
**inner join genre on genre.genreid = dvd.genreid**

**inner join rating on rating.ratingid = dvd.ratingid**

**inner join moviepersonrole on moviepersonrole.dvdid = dvd.dvdid**

**inner join movieperson on moviepersonrole.personid = movieperson.personid**

**inner join role on role.roleid = moviepersonrole.roleid**



**select dvd.dvdtitle, genre.genrename, rating.ratingname, movieperson.personfirstname, movieperson.personlastname, role.rolename**

**from dvd**

**inner join genre on genre.genreid = dvd.genreid**

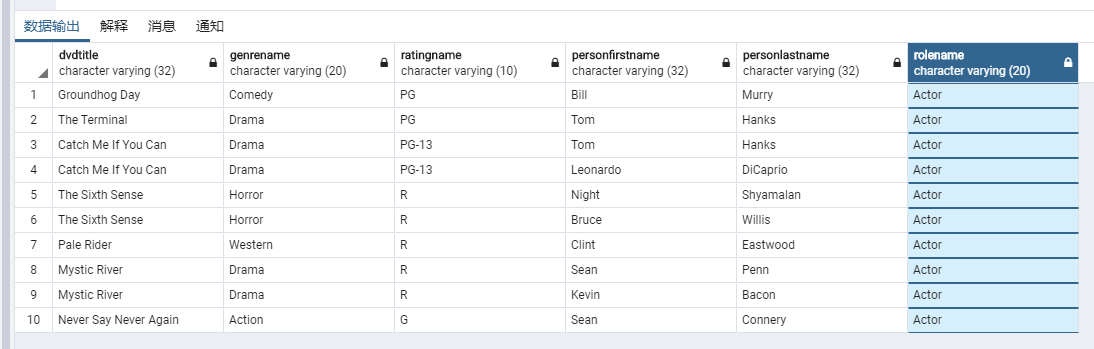
**inner join rating on rating.ratingid = dvd.ratingid**

**inner join moviepersonrole on moviepersonrole.dvdid = dvd.dvdid**

**inner join movieperson on moviepersonrole.personid = movieperson.personid**

**inner join role on role.roleid = moviepersonrole.roleid**

**where rolename = 'Actor'**



1. [ Learning objectives: IS NULL, composite restrictions, subqueries]  
   List all the Members and the DVD Copies they have currently rented and not returned using a subquery. A currently Rented DVD is where the RentalReturnedDate is NULL.
   * Genre and Rating should be the names, not IDs.
   * The query result should show the Member’s name, DVDTitle, Genre, Rating, a director if one exists in the database, DVD Copy, and the request and ship dates for each DVD:

**Note** that in the test data set the movie “Never Say Never Again” is in the Rental table yet has not been shipped. You need to account for this.

Based on the definition, I fixed stock\_status first; changed someone to be false because their RentalReturnedDate is NULL. Still, this can be done automatically with triggers and functions but I just left them to be manually so far. Therefore,

select member.memberfirstname, member.memberlastname, dvd.dvdtitle, genre.genrename, rating.ratingname,

CONCAT(movieperson.personfirstname, ' ', movieperson.personlastname) as Director,

dvd\_copy.copyid, rental.rentalrequestdate, rental.rentalshippeddate

from rental

inner join member on rental.memberid = member.memberid

inner join dvd on rental. dvdid = dvd.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join rating on rating.ratingid = dvd.ratingid

inner join moviepersonrole on moviepersonrole.dvdid = dvd.dvdid

inner join movieperson on moviepersonrole.personid = movieperson.personid

inner join role on role.roleid = moviepersonrole.roleid

inner join dvd\_copy on rental.rentalid = dvd\_copy.rentalid

where rental.RENTALSHIPPEDDATE is not null and rental.RENTALRETURNEDDATE is null and role.roleid=3



The movie “Never Say Never Again” has not been shipped yet so the composite restrictions set as to find which one’s rental shipped date is not null and also rental returned date is null so that we can know which one are currently rented DVD. Also, “role.roleid=3” means director and “CONCAT(movieperson.personfirstname, ' ', movieperson.personlastname) as Director” gives the combination of the name.

1. [ Learning objectives: Correlated sub-queries]

List all the customers and the most recent DVD which they have rented, including the title, Genre, Rating, DVD copy and date of rental.

* Consider using the TOP restriction in MSSQL or the ROWS restriction in Oracle within your correlated sub-query.

Because I was using PostgreSQL this time, I used the way of combination by “distinct on ()” and “where rental.rentalrequestdate < now()” instead of TOP or COWS in other DBMS. Therefore, I did deduplication my data based on the user name and comparation with current timestamp function NOW().

select distinct on (member.memberfirstname, member.memberlastname) member.memberfirstname, member.memberlastname, dvd.dvdtitle, genre.genrename, rating.ratingname, dvd\_copy.copyid, rental.rentalrequestdate

from rental

inner join member on rental.memberid = member.memberid

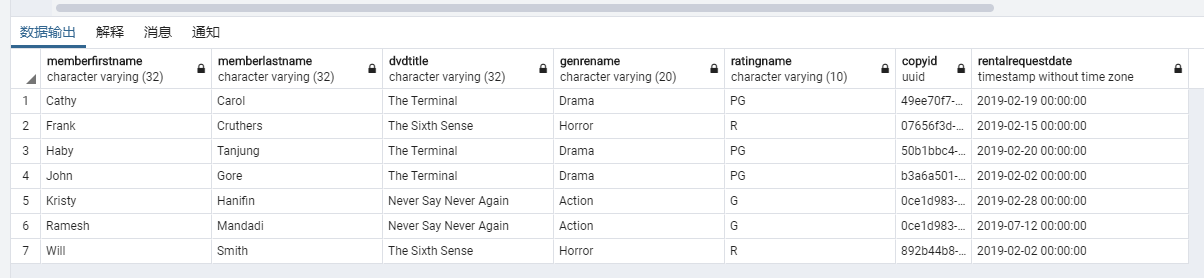
inner join dvd on rental.dvdid = dvd.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join rating on rating.ratingid = dvd.ratingid

inner join dvd\_copy on dvd\_copy.dvdid = dvd.dvdid

where rental.rentalrequestdate < now()



Part 3: Aggregates and Grouping

1. [ Learning objectives: aggregates, joins, GROUP BY]  
   Write a query to list the DVD titles and how many times each one has been rented. The result should be DVD Title, Genre, Rating and number of rentals for each DVD.

Generally, I just count the times of the rental for each DVD by using COUNT function based on rental then sort by their dvdid in the group.

select dvd.dvdtitle, genre.Genrename, rating.Ratingname, count(rental.rentalid) as times

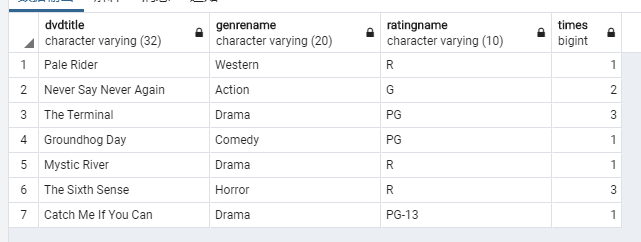
from rental

inner join dvd on dvd.dvdid = rental.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join rating on rating.ratingid = dvd.ratingid

group by dvd.dvdtitle, genre.Genrename, rating.Ratingname

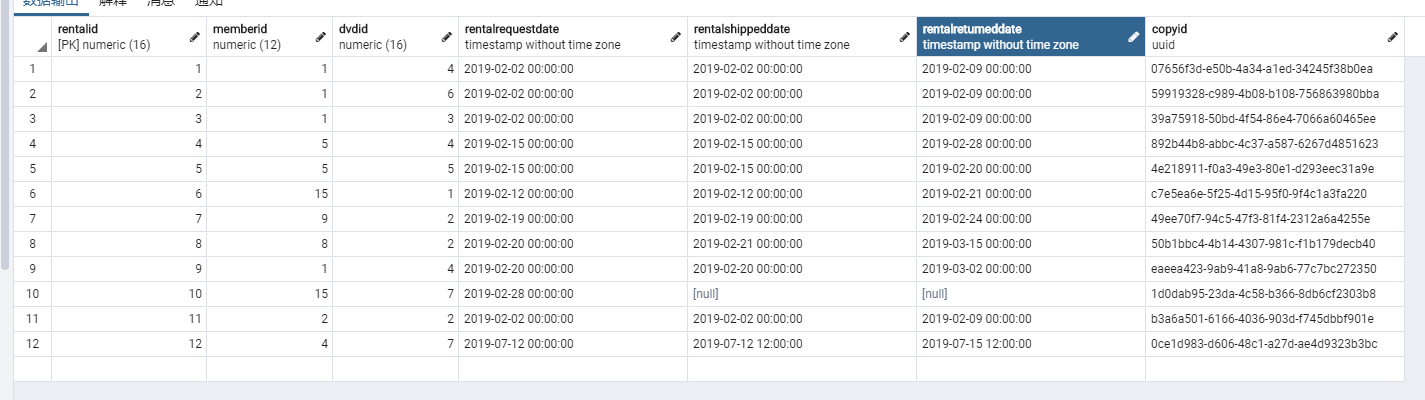


1. [ Learning objectives: aggregates, date differences, grouping]  
   Write a query that returns the average number of days that a DVD is out. The query should return the DVDID, DVD Title, Genre, Rating and the average number of days rented.

Hints/Notes:

* + Look at the Date Difference functions in Oracle and MSSQL to determine the number of days a DVD was out.
  + Think about what you need to GROUP on, and what you will need to AVG.
  + You may need to update the rental table and return a few DVDs (update return date) to make the data a bit more interesting.
  + Genre and Rating should be their names, not IDs.

First, I updated some of data which about their return date for a bit more interesting.



SELECT dvd.dvdid, dvd.DVDTitle, genre.Genrename, rating.Ratingname, AVG(DATE\_PART('day', rentalreturneddate - rentalshippeddate))

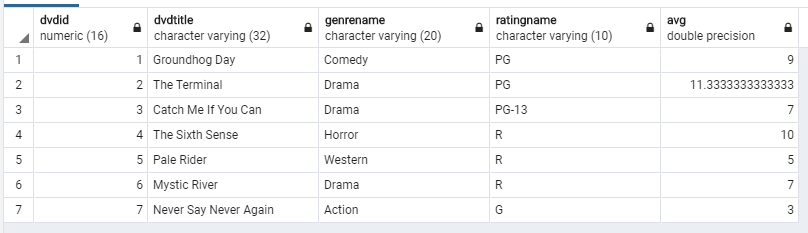
from rental

inner join dvd on dvd.dvdid = rental.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join rating on rating.ratingid = dvd.ratingid

group by dvd.dvdid, dvd.DVDTitle, genre.Genrename, rating.Ratingname order by dvd.dvdid



**Generally, I used date difference by using DATE\_PART() function in PostgreSQL to count numbers of difference in days. After that, I counted the average number of days rented by AVG function. Finally, ordering the results by DVD Id.**

**Additionally, it was interesting to warm up the knowledge about data type of NULL value for “Never Say Never Again” will been counted as 0.**

1. [ Learning objectives: aggregates, TOP, Filtering within Having clause, CTE/Subquery]  
   Utilize common table expression (CTE) as part of your solution to write a query that returns the three most popular movie Genres based on the number of rentals within the last 5 years. The query should return the Genre and the number of DVDs rented for the genre with the most rentals.

WITH cte\_genre\_amounts (genrename, count) AS (

select genre.genrename, count(rental.rentalid)

from rental

inner join dvd on rental.dvdid = dvd.dvdid

inner join genre on genre.genreid = dvd.genreid

where DATE\_PART('year', now()) - DATE\_PART('year', rentalrequestdate) <=5

group by genre.genrename

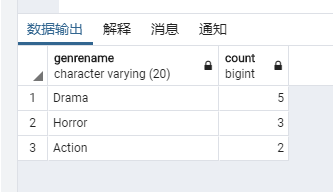
)

SELECT

genrename, count

FROM cte\_genre\_amounts

order by count desc limit 3



**Query for filter data from last 5 years:**

**where DATE\_PART('year', now()) - DATE\_PART('year', rentalrequestdate) <=5**

**Query for count: count(rental.rentalid)**

**Group by genre.genrename**

# Part 4: OLAP and Grouping Functions

1. [ Learning objectives: RANK, GROUP BY]   
   List membership types by rank based on the number of movies that have been rented. You should have two query results, one rank, and one using dense rank.
   * Think about what you need to GROUP on, and what you will need to COUNT.
   * Look at the RANK function, for example in the Oracle Reference Manual.

**I feel a little confused about the meaning of question which “List membership types by rank based on the number of movies that have been rented.”**

**First, if I just simply rank the membership types we have, it will not show any different.**

with cte\_member\_rented (type, rented) as (

select membership.membershiptype, count(rental.rentalid) as rented

from rental

inner join member on rental.memberid = member.memberid

inner join membership on member.membershipid = membership.membershipid

group by member.memberid, membership.membershiptype)

select type,

rank() over (order by sum(rented)) as rank,

dense\_rank() over (order by sum(rented)) as "Dense Rank"

from cte\_member\_rented

group by type



**This result makes me think I misunderstand about what this question want me to do. Therefore, I left my “extra” answer for ranking members based on how many DVD they rented with their membership type. I just to state that I understand how and what rank functions do by this.**

with cte\_member\_rented (customer, type, rented) as (

select concat(member.memberfirstname, ' ', member.memberlastname), membership.membershiptype, count(rental.rentalid) as rented

from rental

inner join member on rental.memberid = member.memberid

inner join membership on member.membershipid = membership.membershipid

group by member.memberid, membership.membershiptype)

select customer, type,

rank() over (order by sum(rented)) as rank,

dense\_rank() over (order by sum(rented)) as "Dense Rank"

from cte\_member\_rented

group by customer, type



Therefore, the company maybe can use this to present a best consumer award or something else. Anyway, this will give us a clear shot about the difference between rank and dense rank.

1. [ Learning objectives: ROLLUP, CUBE]  
   Write a query that returns the total revenue collected based on each membership type for each DVD. The Query should include grand totals.

Review ROLLUP and CUBE.

I am confused on the meaning of this question; however, I saw the statement on the discussion board at the blackboard. According to what your said, “for example, instead of DVD, it might be interesting to see Movie Genre, meaning which Genre is more popular with which plan.” I created two ways which using ROLLUP and CUBE to check the preferences for each genre in each membership type.

select distinct on (genre.genrename, membership.membershiptype) genre.genrename, membership.membershiptype, count(rental.rentalid) as Preferences

from rental

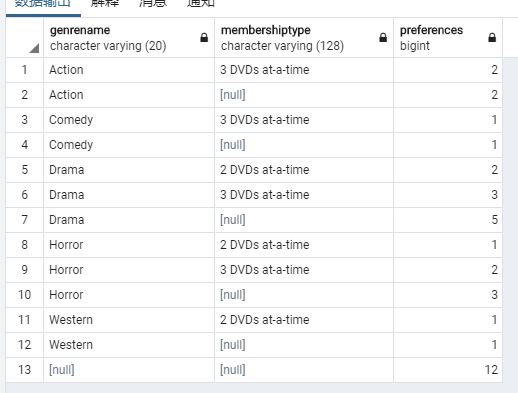
inner join dvd on rental.dvdid = dvd.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join member on rental.memberid = member.memberid

inner join membership on member.membershipid = membership.membershipid

group by ROLLUP(genre.genrename, membership.membershiptype)



select distinct on (genre.genrename, membership.membershiptype) genre.genrename, membership.membershiptype, count(rental.rentalid) as Preferences

from rental

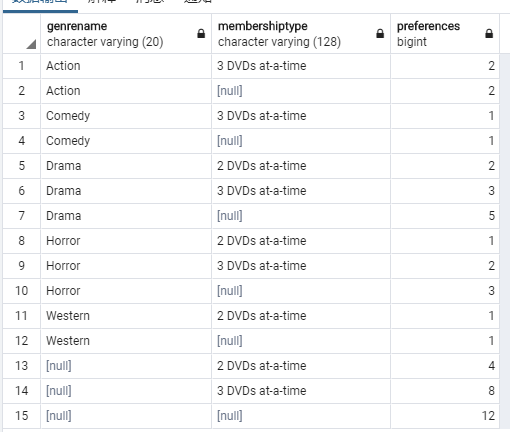
inner join dvd on rental.dvdid = dvd.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join member on rental.memberid = member.memberid

inner join membership on member.membershipid = membership.membershipid

group by Cube(genre.genrename, membership.membershiptype)



Thus, based on what we saw from the output, we will know the different preferences for different member type owners. I hope this answer has same functionality with what this question want me to show.

# Part 5: Extra Credit Problems

1. [ Learning objectives: PIVOT]

Write a query that lists the number of DVDs rented for each Genre and Year. Then PIVOT this so that each Genre is listed as a column heading with one row for each year.

I failed on this one. I thought it should be like this:

SELECT \*

FROM crosstab(

'SELECT date\_part('year',rental.rentalrequestdate), genre.genrename, count(rental.rentalid)

FROM rental

inner join dvd on dvd.dvdid = rental.dvdid

inner join genre on genre.genreid = dvd.genreid

ORDER BY 1,2'

) AS pvt\_year ("Year" int, "Action" int, "Adventure" int, "Comedy" int, "Crime" int, "Drama" int,

"Epics" int, "Musicals" int, "Science Fiction" int, "War" int, "Western" int, "Romance" int, "Horror" int);

**But I keep receiving error called “ ERROR: syntax error at or near "year" LINE 3: 'SELECT date\_part('year',rental.rentalrequestdate), genre...”**

**SQL : 42601**

However, the way to catch year from timestamp in POSTGRESQL should be the way I used. Hope I did some parts of Pivot is right in here.

1. [ Learning objectives: Extracting month from date, aggregates, GROUP BY]  
   List each Genre and the Month that genre has the most rentals over the past three years. For example, your query should return that the Romance genre was the most popular in February of the year of your choice, with 850 Romance DVDs rented. The query should return the Genre, Month, year, and the number of DVDs rented for the genre with the most rentals for that month and year.
   * Look at functions extracting Month from the rental date.
   * Think about using subqueries, you can also practice common table expressions if using SQL Server.

In POSTGRESQL, DATE\_PART(field,source) function is used to retrieve subfields e.g., year, month, week from a date or time value. Therefore, I did below.

select genre.genrename, date\_part('month', rental.rentalrequestdate) as month, date\_part('year', rental.rentalrequestdate) as year,

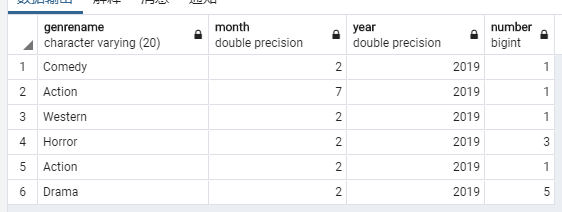
count(rental.rentalid) as number

from rental

inner join dvd on rental.dvdid = dvd.dvdid

inner join genre on dvd.genreid = genre.genreid

group by (genre.genrename, date\_part('month', rental.rentalrequestdate), date\_part('year', rental.rentalrequestdate))



Although the dataset are just what has original one, we can see there are on the right way of the target for this question.

Conclusion

After I finished this assignment, I feel I forgot and recovered a lot of knowledge from what I learned from CS 579 last semester. Something I wrote might be wrong or misunderstood in this work because I admit that this homework is not quite simple, but I was fairly satisfactory for almost things. I also think I need to recover some knowledge about the trigger functions because I wanted to write some of them, but I gave up at last because I failed.

Moreover, I apologize for my pgAdmin UI language. Let me know if you feel bad on it and I will try to change it.

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

Save your assignment as ***lastnameFirstname\_assign1.1.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.