Mi Gao

migao@bu.edu

Assignment 2.0 Performance Tuning

MET CS 779

MET CS 779 Assignment 2.0

Table of Contents

[1. Part A 3](#_Toc46176183)

[1) Document what DDL changes you need to make and provide the SQL and the screenshots showing the changes being made. 3](#_Toc46176184)

[2) Document the record count from the loaded tables by providing the screenshot. 3](#_Toc46176185)

[2. Part B Question 1 3](#_Toc46176186)

[3) What changes would you make to the provided schema so that it can scale to handle a million or more transactions per day? Identify the scalability-limiting schema features and propose changes that would make them scalable. 4](#_Toc46176187)

[4) Would you denormalize? If you would denormalize, how would you maintain the denormalizations? Justify your design changes regardless of your decision about whether to denormalize. 4](#_Toc46176188)

[5) Please provide parts of the ERD design changes you are proposing with your suggested changes and paste it below. 4](#_Toc46176189)

[3. Part B Question 2 4](#_Toc46176190)

[6) How would you design a denormalized RentalHistory table to reduce the number of tables that need to be joined? 4](#_Toc46176191)

[7) How would you maintain this data, meaning how and when would it be populated and updated? Compare and contrast some different approaches (think stored procedures vs. triggers). 4](#_Toc46176192)

[8) **Provide an ERD design** for the RentalHistory table with your suggestions and the explanations of your changes including an ERD of the RentalHistory table below. 5](#_Toc46176193)

[4. Part C 5](#_Toc46176194)

[1) Below query is used to identify DVDs that are similar to the DVDs that the member has rented, in this case based on the director of the movie. 5](#_Toc46176195)

[2) Queries like the following are run frequently to identify DVDs that a member has not rented. Note that the MemberID is passed from the application layer and may be different each time the query is run. 5](#_Toc46176196)

[3) The following view assembles the data for a DVD in a form useful for output to a user interface, so it is used extensively by the application. 5](#_Toc46176197)

[4) Review your solution to question 7 from Programming Assignment Part 1. Customer is reporting that it is running slow. Recall the question asked to 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. The attributes returned were 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. The Customer just cares about which movies have been rented and not returned at minimum. 6](#_Toc46176198)

[5) Review your solution to question 9 from Programming Assignment Part 1. Customer is reporting that it is running slow. Recall that the question asked for 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. 6](#_Toc46176199)

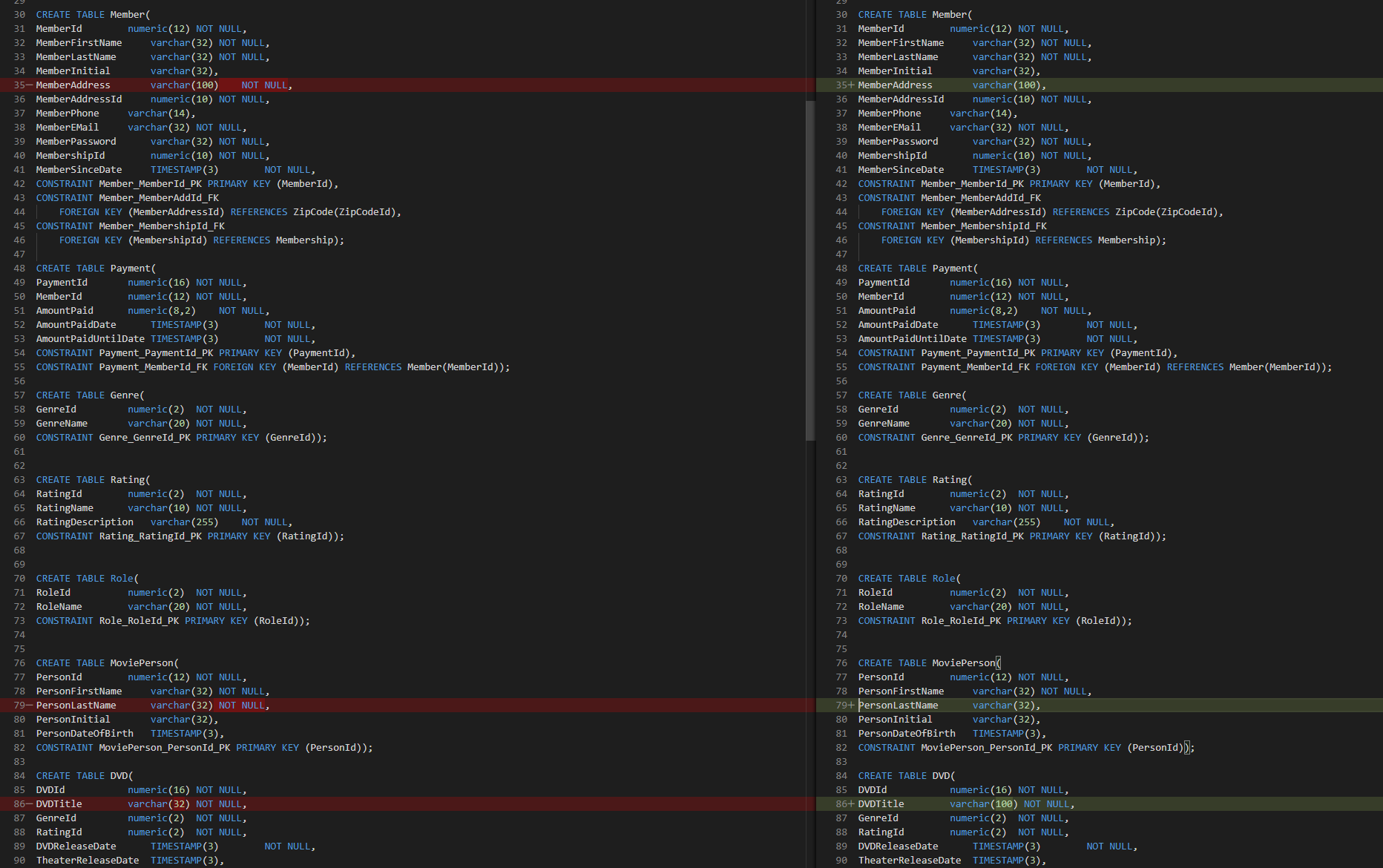
[6) There are many ways to improve the scalable performance of the following stored procedure, which is running increasingly slowly as the database grows. Identify the changes to improve the scalable performance of this stored procedure. 6](#_Toc46176200)

[5. Conclusion 7](#_Toc46176201)

[5. Revision History 8](#_Toc46176202)

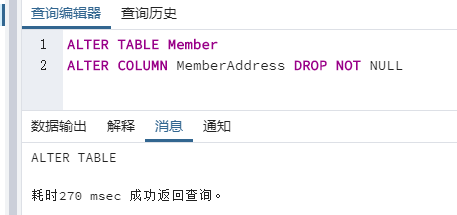
# Part A

## Document what DDL changes you need to make and provide the SQL and the screenshots showing the changes being made.



Member table:

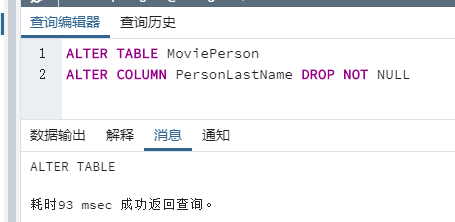
MemberAddress varchar(100) NOT NULL 🡪 MemberAddress varchar(100)





MoviePerson table:

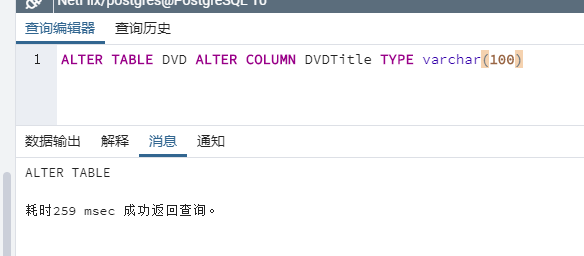
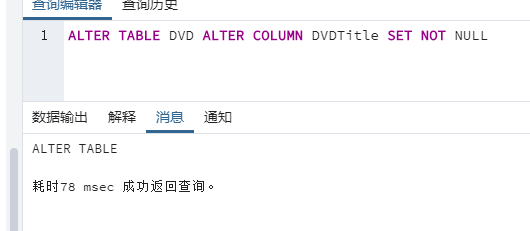
PersonLastName varchar(32) NOT NULL 🡪 PersonLastName varchar(32)





DVD table:

DVDTitle varchar(32) NOT NULL 🡪 DVDTitle varchar(100) NOT NULL



And for indexs, there has a new unique index been created in the original one:

CREATE UNIQUE INDEX I\_DVDTitle ON DVD(DVDTitle);



Also I need to drop extra column I made in last assignment for run script in rental table successfully.



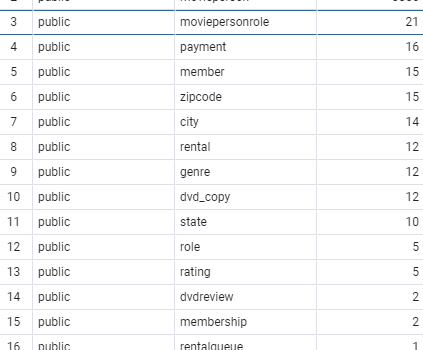
## Document the record count from the loaded tables by providing the screenshot.

I did not see the way like Oracle and SQL SERVER to execute sql files in once so I just do separately. And I use the query and function to count how many tuples I have after I inserted.

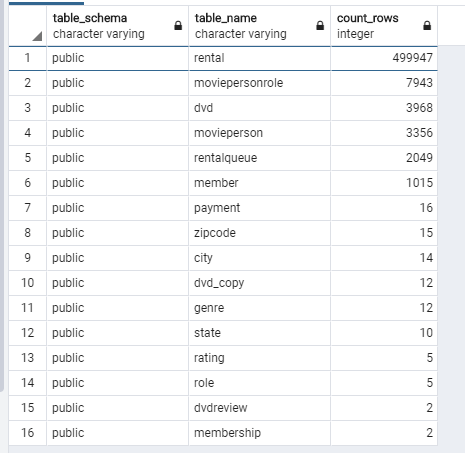




Before I insert new data from scripts:



After inserted:



# Part B Question 1

## What changes would you make to the provided schema so that it can scale to handle a million or more transactions per day? Identify the scalability-limiting schema features and propose changes that would make them scalable.

If we talk about scalability, the first thing I thought will be denormalization. If I may add some indexes to handle such lots of transactions each day. Also, denormalizing some tables to avoid joining multiple tables to retrieve data will be good. After indexing attributes, data between millions of records will be retrieved faster. In my opinion, I need to access several tables in the original schema to retrieve data, so indexing required attributes and denormalizing some tables into large tables is essential to improve scalability.

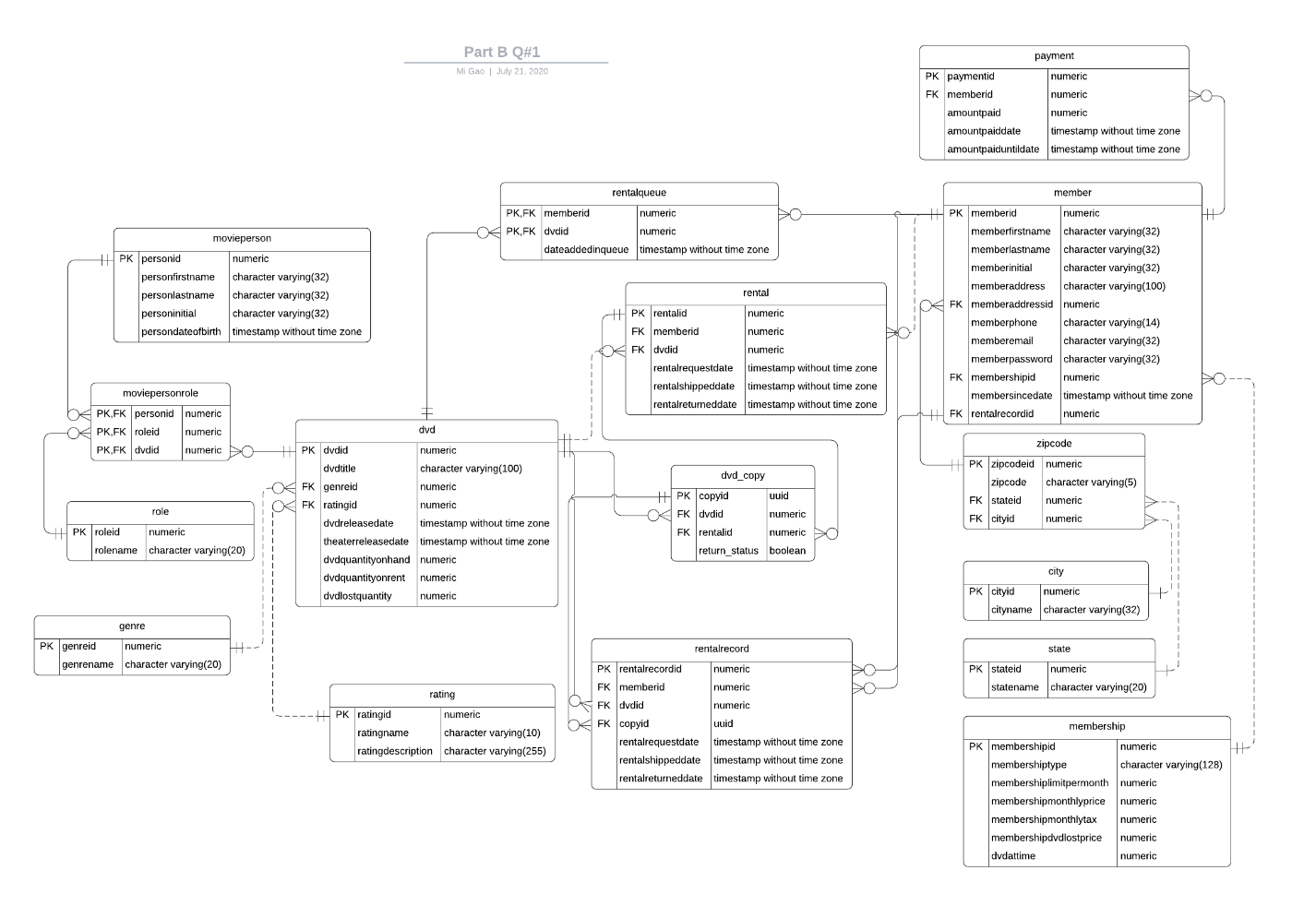
Although performance is reduced when adding or updating data, indexes can speed up searches and queries. So I want to index the following fields: frequently searched fields, sorted fields, and fields in other tables that are joined to multiple table queries; in which means, those foreign keys are the main target to set the index. Therefore, the keys in tables such as dvd, dvd\_copy, member and payment without rentalqueue will be nice because users may usually modified their wish list. So that, I think when query for the payment, the rental records, and monthly billing of each customer’s credit card can be faster at least.

## Would you denormalize? If you would denormalize, how would you maintain the denormalizations? Justify your design changes regardless of your decision about whether to denormalize.

In my opinion, I will denormalize for sure if I want to improve my performance. So far, if I want to query the rental history, the best and fastest way might be put them all together. So that I think I need to put at least dvdid, memberid, member’s name and dvd’s copyid to get there to help user to quickly query the rental. Last time, I combined copyid into rental table as foreign key; however, I dropped that column for insert/load new data successfully. Anyway, when we need query rental information about a member, in this case, we can easily retrieve the information by joining a few tables.

## Please provide parts of the ERD design changes you are proposing with your suggested changes and paste it below.

Notice: I imported CSV from my PostgreSQL schema on Lucidchart and kept working on it so that why this show like this format.



# Part B Question 2

## How would you design a denormalized RentalHistory table to reduce the number of tables that need to be joined?

I find this question I just did in above for improve the speed of querying rental record so that I will do this based on it in this question. I may just add few more things on it if we need such as the director and genre. Therefore, I will not repeat same paragraphs for saving your reading time. I will introduce what we have in this table.

|  |  |
| --- | --- |
| rentalhisoryid | Primary key for the table |
| memberid | Foreign key to member table for member infomation |
| dvdid | Foreign key to dvd table for dvd name |
| personid | Foreign key to movieperson table for the director |
| genreid | Foreign key to genre for genre name |
| copyid | Foreign key to dvd\_copy for the UUID |
| rentalrequestdate | Request date |
| rentalshippeddate | Shipped date |
| rentalreturneddate | Returned date |

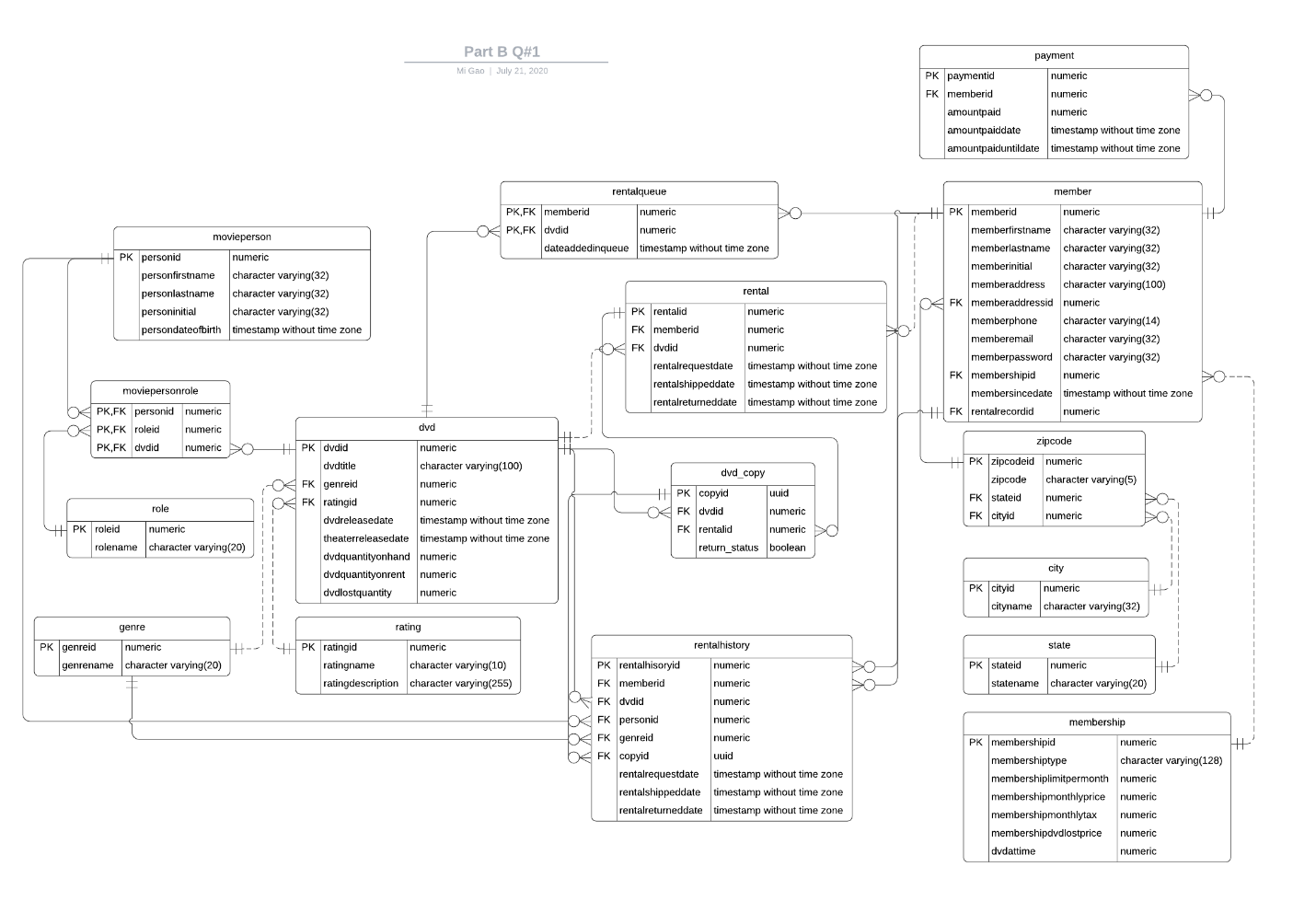
According to this table, we can get all things we need faster then go through the relationships made by normalization when review our rental history.

## How would you maintain this data, meaning how and when would it be populated and updated? Compare and contrast some different approaches (think stored procedures vs. triggers).

When a business processes multiple tables at the same time, it is more appropriate to use a stored procedure. The use of stored procedures will improve performance in general, because the database optimizes the data access plan of the stored procedures and applies the cache to facilitate future checks. Any request for the DVD can create tuple to record.

Also, I would like to use trigger to update the values in the table. For example, when insert or update a value in rental such as the return date, the values in rental history will been created or updated; then, we will know anything about this record. And function and triggers have better abilities on error control and setting conditions.

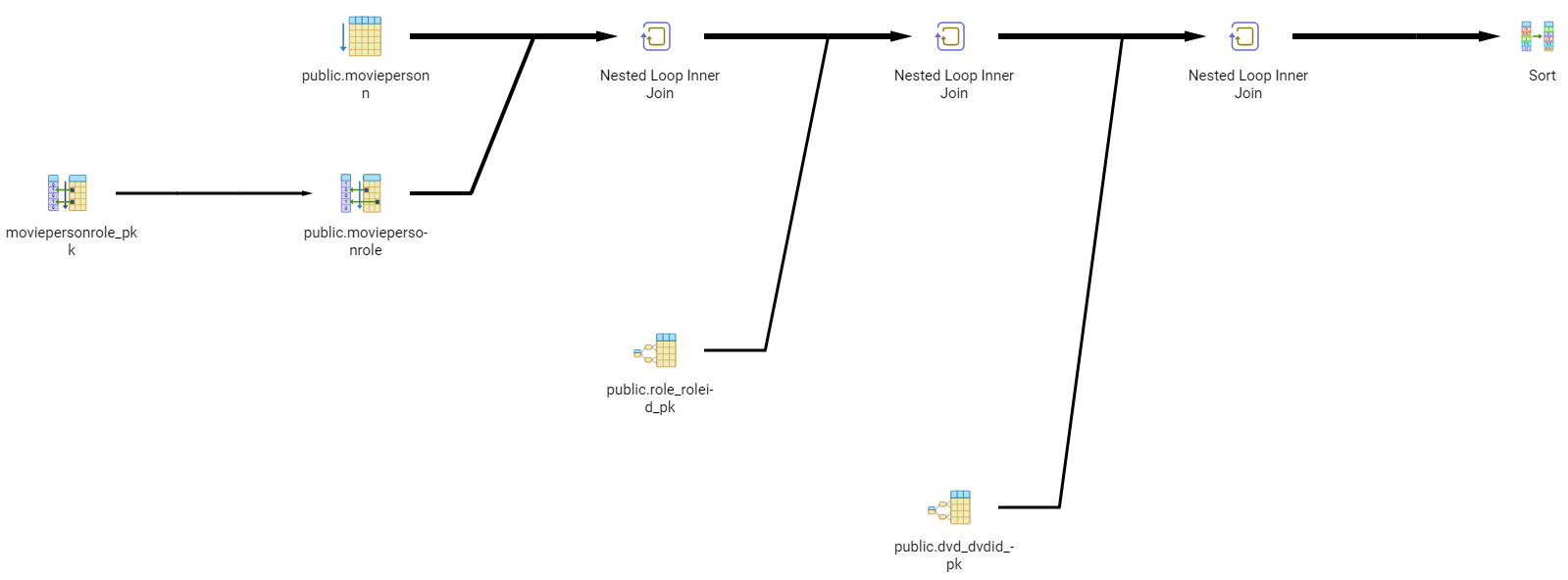
## **Provide an ERD design** for the RentalHistory table with your suggestions and the explanations of your changes including an ERD of the RentalHistory table below.

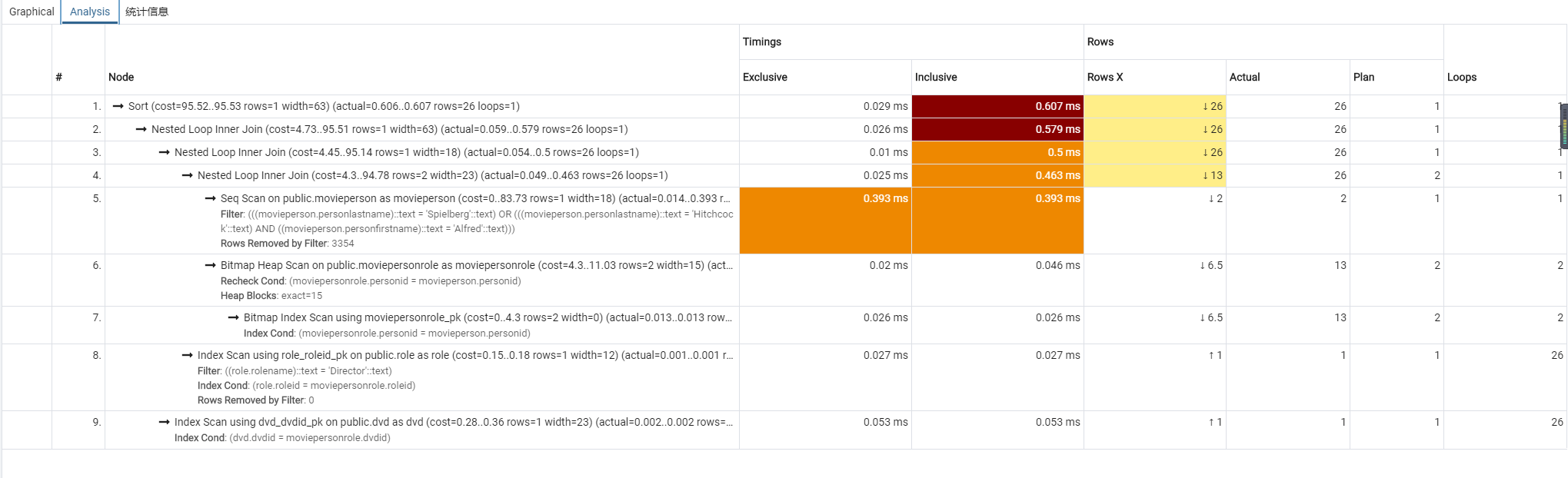


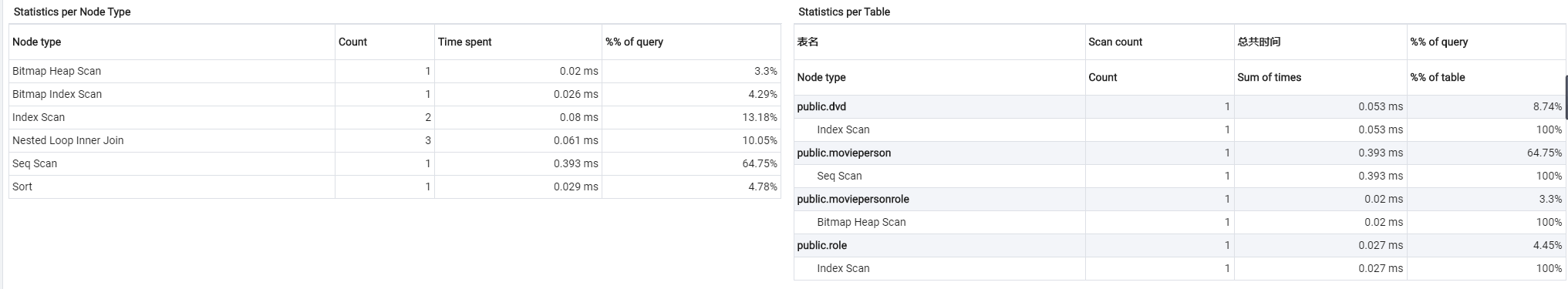
# Part C

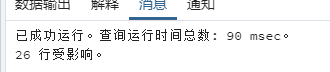
## Below query is used to identify DVDs that are similar to the DVDs that the member has rented, in this case based on the director of the movie.

* 1. What performance issues do you see with this query? Examine the execution plan to confirm your concerns. Are you seeing something within the execution plan which you did not consider?









Runtime: 90ms

I see the I/O operations such as scan and sorting spent most of time. This is a simple filter so the result I think is good enough. This should be the benefits of the index we have in the movieperson. However, maybe change some on JOINs can make all be faster; also, the conditions in WHERE.

* 1. Your recommendation to improve performance:

In my opinion, the performance may be improve by adding index on dvd table; which means, we can add index for dvdtitle again but not the unique key because the contents of movieperson, role, and DVD are not frequently updated or inserted into new entries after all.

Such as the index we have for movieperson:

CREATE INDEX I\_PersonName ON MoviePerson(PersonFirstName,PersonLastName);

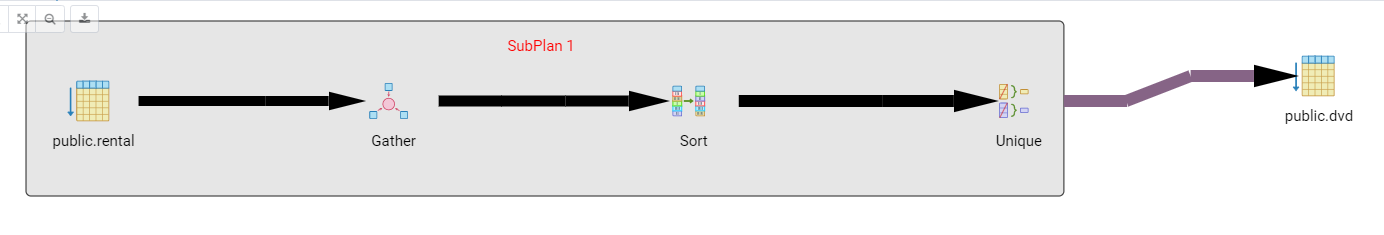
the performance may be improve by adding index on dvd table; which means, we can add index for dvdtitle again but not the unique key.

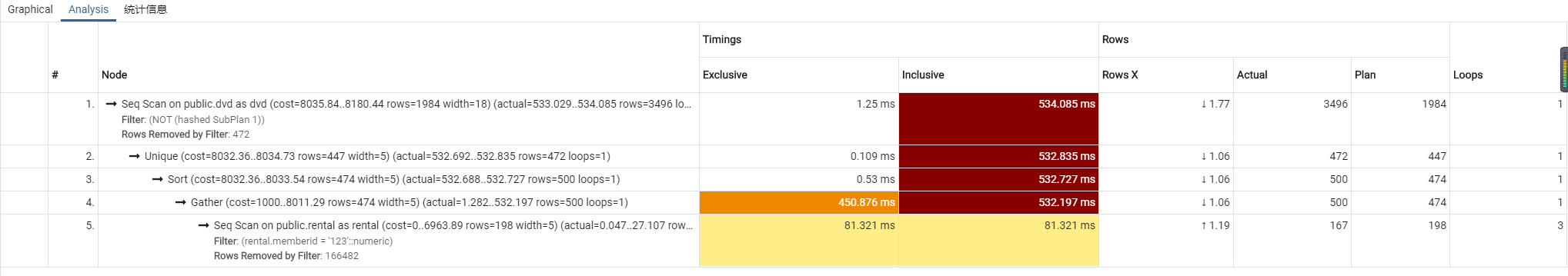
Also, create index such below against the query in this question may also improve the speed I think:

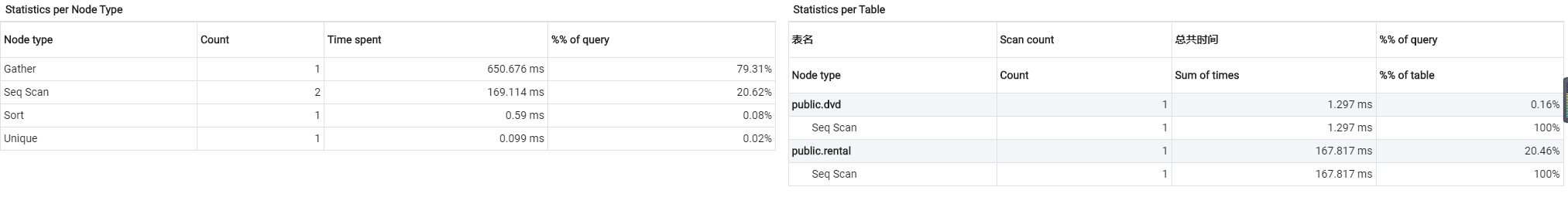
CREATE INDEX people\_names ON MoviePerson ((PersonFirstName || ' ' || PersonLastName));

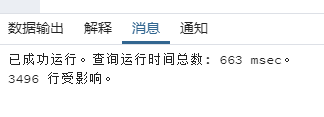
## Queries like the following are run frequently to identify DVDs that a member has not rented. Note that the MemberID is passed from the application layer and may be different each time the query is run.

* 1. What performance issues do you see with this query? Examine the execution plan to confirm your concerns. Are you seeing something within the execution plan which you did not consider?









Runtime: 663ms.

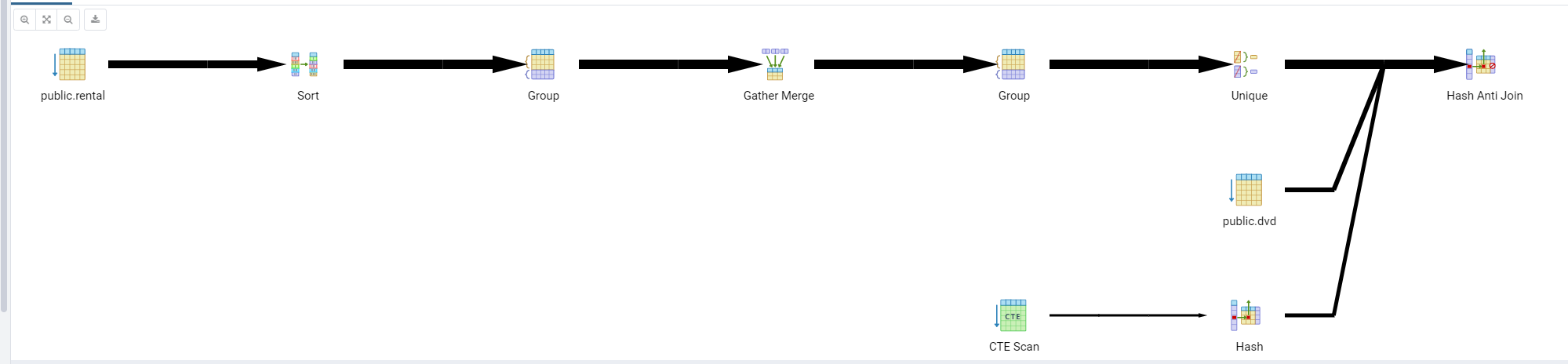
When I see this inclusive running time I did not consider at first. I thought this might be caused by NOT IN. This may cause the full scan for all 3496 tuples.

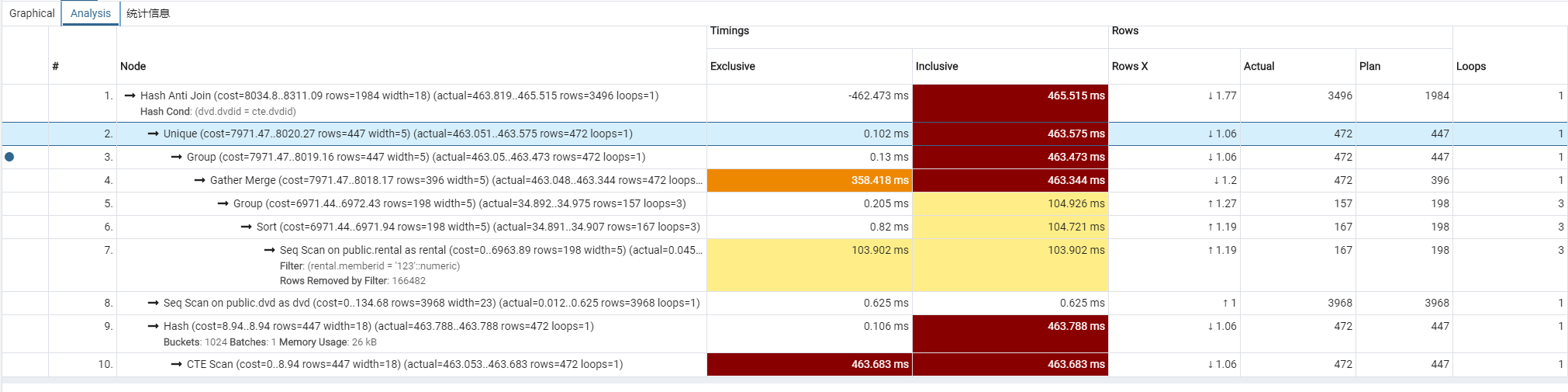
* 1. Your recommendation to improve performance:

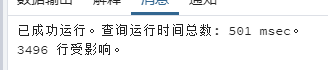
I saw subquery question then immediately remember what I learn in last assignment which CTE can improve performance. I would like to use LEFT JOIN instead of NOT IN so that I just make a test.

1. with cte as (
2. SELECT DISTINCT DVDId FROM Rental WHERE MemberId = 123
3. group by dvdid
4. )
5. SELECT DVDTitle AS "DVD Title" FROM DVD
6. left join cte on dvd.dvdid = cte.dvdid

where cte.dvdid is null





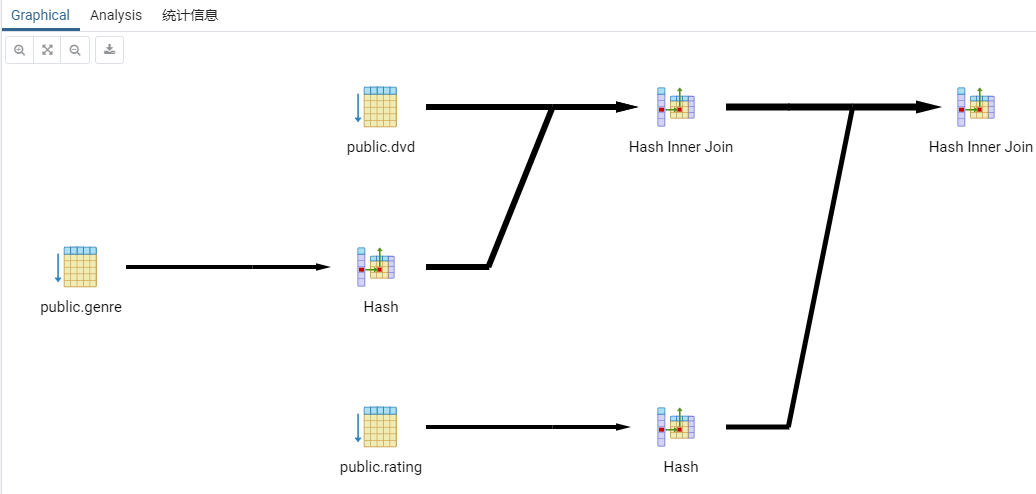


Runtime: 501ms.

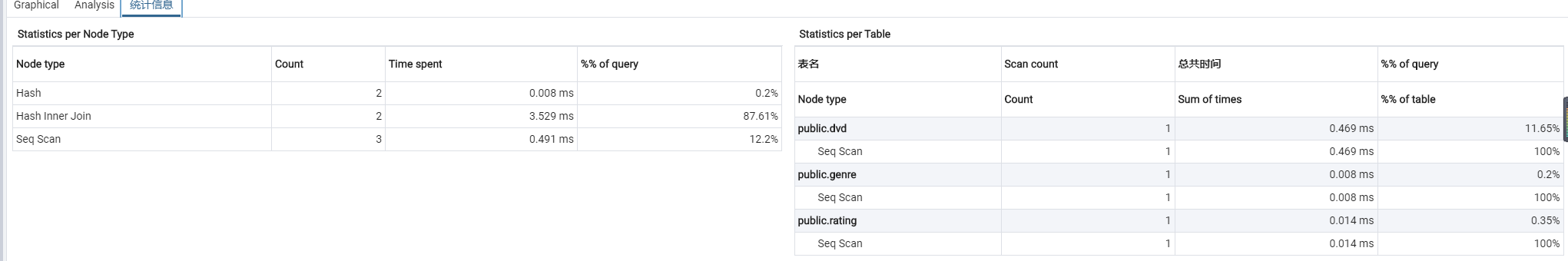
I tried to run several times and the average around 500 to 600ms. I think it is can be counted as a little improvement.

## The following view assembles the data for a DVD in a form useful for output to a user interface, so it is used extensively by the application.

* 1. What performance issues do you see with this query? Examine the execution plan to confirm your concerns. Are you seeing something within the execution plan which you did not consider?

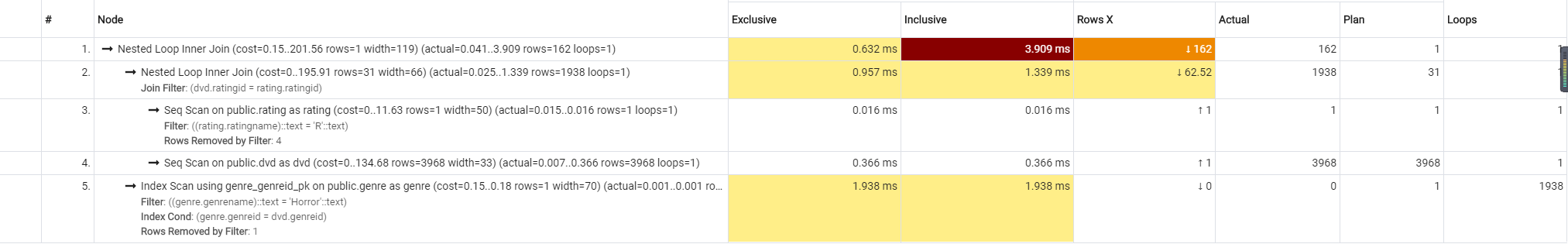


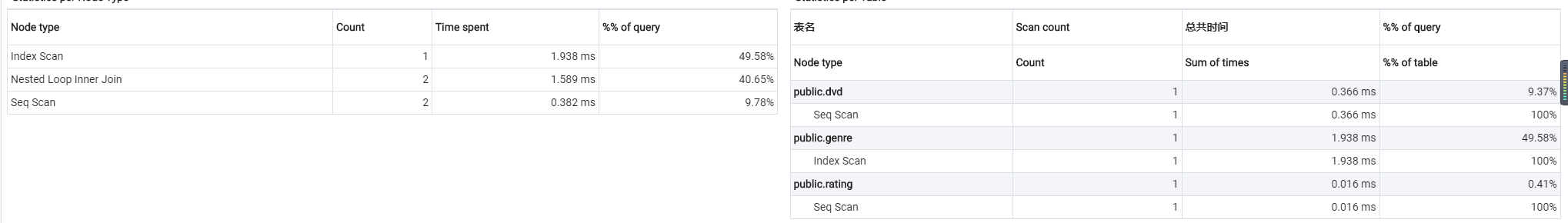




When do query such:

SELECT \* FROM DVDView WHERE Genre = 'Horror' AND Rating = 'R';





* 1. Your recommendation to improve performance:

I do not really have some good idea of improvement for the queries. However, if against to what I used on example query:

SELECT \* FROM DVDView WHERE Genre = 'Horror' AND Rating = 'R';

My suggestion is that we maybe create some indexes for foreign keys in DVD table for such as genre and rating because they will always being searched. After all, the key in dvd table looks will not be updated usually.

Create index i\_genre on dvd(genreid)

Create index i\_rating on dvd(ratingid)

## Review your solution to question 7 from Programming Assignment Part 1. Customer is reporting that it is running slow. Recall the question asked to 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. The attributes returned were 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. The Customer just cares about which movies have been rented and not returned at minimum.

* 1. What performance issues do you see with this query? Examine the execution plan to confirm your concerns. Are you seeing something within the execution plan which you did not consider?

This takes me more than 2 seconds for currently large size of rental table. However, if the customer only just cares about which movies have been rented and not returned at minimum. I can really reduce a lot of things for full scanning such as movieperson’s name.



* 1. Your recommendation to improve performance:

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

from rental

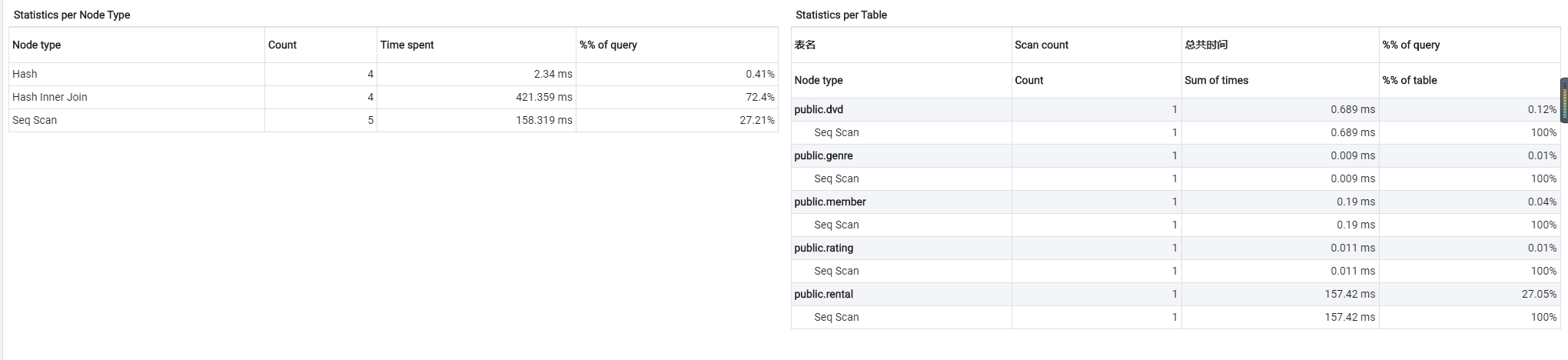
inner join member on rental.memberid = member.memberid

inner join dvd on rental. dvdid = dvd.dvdid

inner join genre on genre.genreid = dvd.genreid

inner join rating on rating.ratingid = dvd.ratingid

where rental.RENTALSHIPPEDDATE is not null and rental.RENTALRETURNEDDATE is null

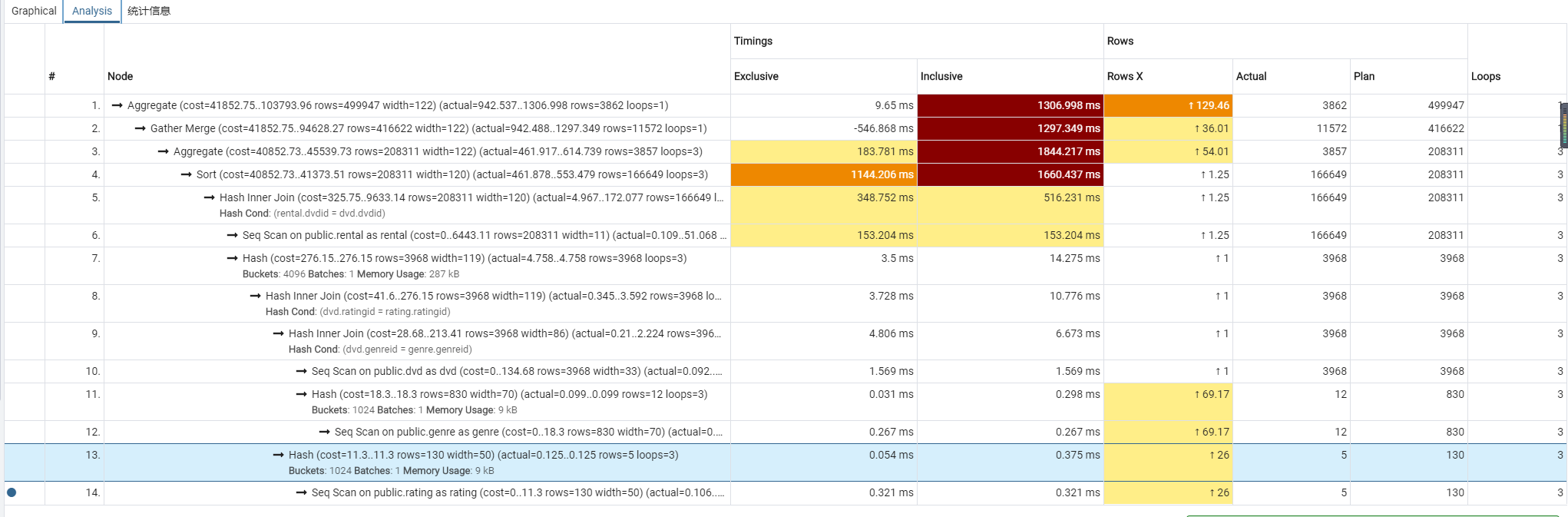




Now, this will take around 600ms to execute and main time was spent on hash joins. And the result can tell customers the DVD information and the return status clearer and quicker.

## Review your solution to question 9 from Programming Assignment Part 1. Customer is reporting that it is running slow. Recall that the question asked for 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.

* 1. What performance issues do you see with this query? Examine the execution plan to confirm your concerns. Are you seeing something within the execution plan which you did not consider?



It takes 1 secs 477 msec.

They did loop 3 times on sorting of rental table with 166649 times. This is a surprise for me; I need to reduce these.

* 1. Your recommendation to improve performance:

I decided to use CTE to package the count of rental times first then do to query. This looks worked.

with cte (dvdid, times) as (select dvd.dvdid, count(rental.rentalid) from rental

inner join dvd on dvd.dvdid = rental.dvdid

group by dvd.dvdid)

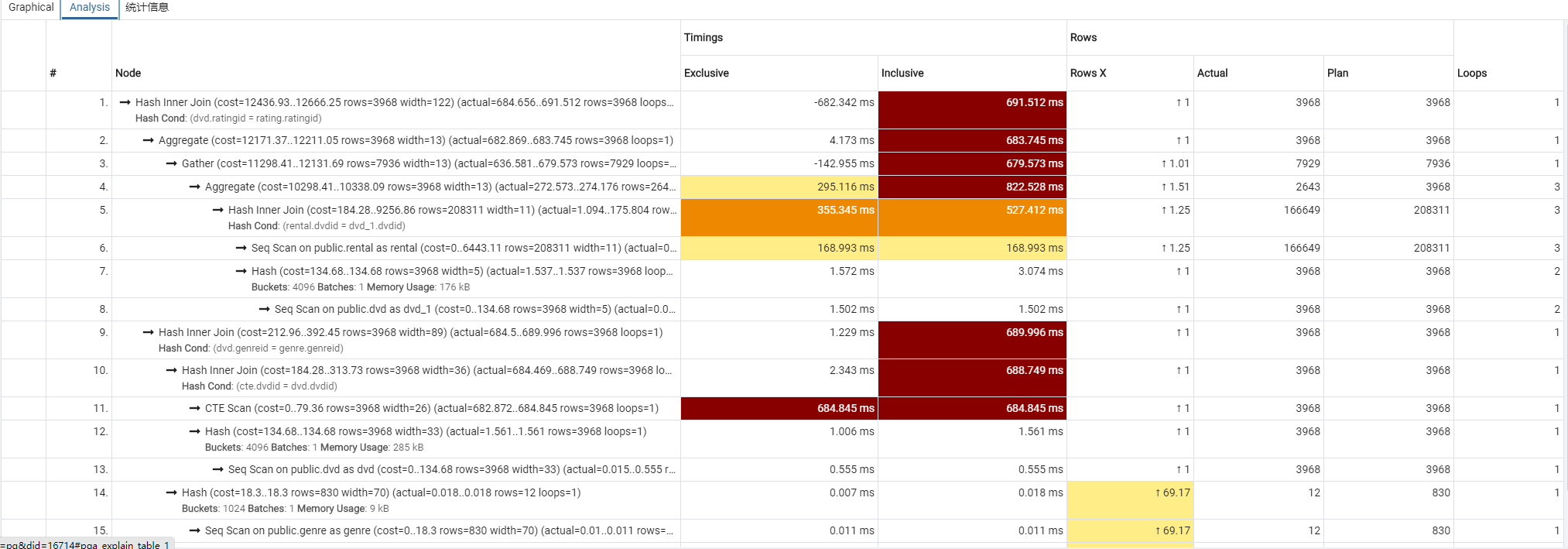
select dvd.dvdtitle, genre.Genrename, rating.Ratingname, cte.times

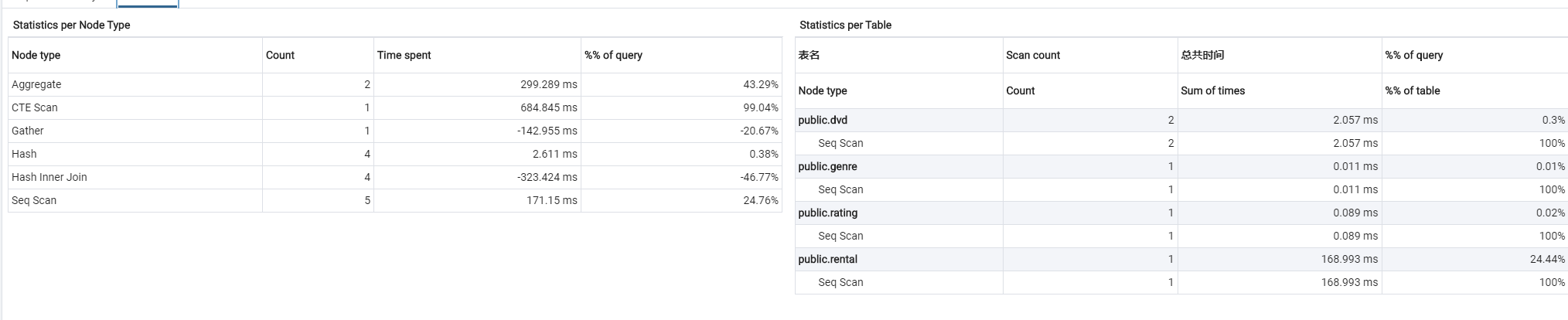
from dvd

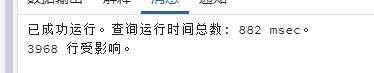
inner join genre on dvd.genreid = genre.genreid

inner join rating on dvd.ratingid = rating.ratingid

inner join cte on dvd.dvdid = cte.dvdid







Now, it only spend half of the time compared with the old version. However, I think there should have the better way to keep improving on this.

## There are many ways to improve the scalable performance of the following stored procedure, which is running increasingly slowly as the database grows. Identify the changes to improve the scalable performance of this stored procedure.

* 1. What performance issues do you see with this stored procedure:

I saw there use function Max():

(SELECT MAX(PaymentId) FROM Payment WHERE MemberId=P\_MemberId);

Also, some of subqueries may be the case to getting longer runtime:

1. -- Get AmountPaidUntilDate
2. SELECT AmountPaidUntilDate INTO V\_UntilDate FROM Payment
3. WHERE MemberId = P\_MemberId
4. AND PaymentId =

(SELECT MAX(PaymentId) FROM Payment WHERE MemberId=P\_MemberId);

* 1. Your recommendation to improve performance:

I have to apologize for this question first. Because of my POSTGRESQL on my computer version is 10 not 11 so that I can not use stored procedure in actual so that I just tried to do some test on online VMs this time. The version upgrade and the resulting data migration are more time-consuming than I thought; especially after importing the new rental table, my computer was down for a while. However, I will write down those improvements and prepare my new version of DBMS before the next coming assignment.

For that part, SELECT INTO fills V\_UntilDate. I am not pretty sure the function MAX() will effect performance too much; however, I do this way which I try to get the same functionality by OUTER JOIN:

left outer join payment p on payment.memberid = p.memberid

where member.memberID = 1 and payment.memberid > p.memberid

I am not sure about I should use member.memberID = 1 or ORDER BY for the top; hope I did right. But anyway, I think use JOINs will helpful to be faster at least.

-- Get AmountPaidUntilDate

select payment.AmountPaidUntilDate into V\_UntilDate

from payment

inner join member on payment.memberid = member.memberid

left outer join payment p on payment.memberid = p.memberid

where member.memberID = 1 and payment.memberid > p.memberid

# Conclusion

I learned a lot and opened my mind by thinking about how to improve the performance in this assignment. Although I was still not very clear about this section, this made me keep changing the way to solve problems in different ways. Also, I think I still not quite familiar with stored procedure.

# 5. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Version** | **Description** |
| Mi Gao | 07/19/20 | 1.0 | Initial Document Creation |
| Mi Gao | 07/21/20 | 1.5 | Added more contents |
| Mi Gao | 07/22/20 | 2.0 | Finishing |