Table of Contents

[Part A: Data Load 1](#_Toc58872293)

[1. Inspect the DDL 1](#_Toc58872294)

[2. Document DDL Changes 1](#_Toc58872295)

[3. Loading the Test Data 3](#_Toc58872296)

[4. Document Record Count 4](#_Toc58872297)

[Part B: Questions 6](#_Toc58872298)

[1. Identify Performance and Other Scaling Problems 6](#_Toc58872299)

[2. Topic Discussions 8](#_Toc58872300)

[Part C: Questions 10](#_Toc58872301)

[1. Query Performance Issue 1 11](#_Toc58872302)

[2. Query Performance Issue 2 13](#_Toc58872303)

[3. Query Performance Issue 3 15](#_Toc58872304)

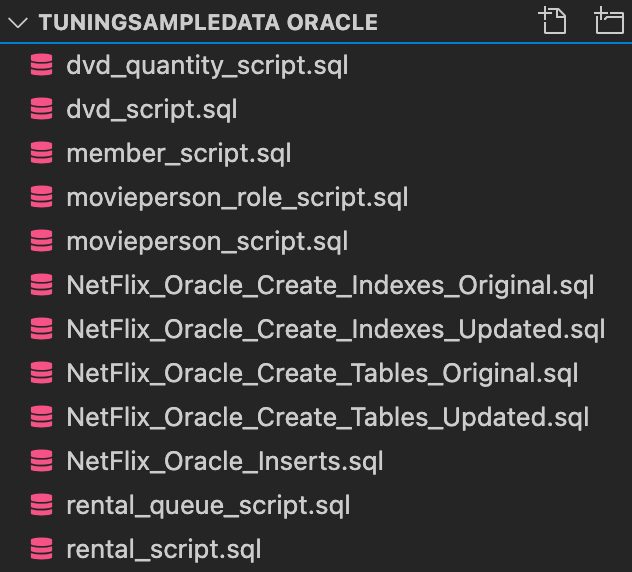
[4. Query Performance Issue 4 17](#_Toc58872305)

[5. Query Performance Issue 5 19](#_Toc58872306)

[6. Query Performance Issue 6 21](#_Toc58872307)

## Part A: Data Load

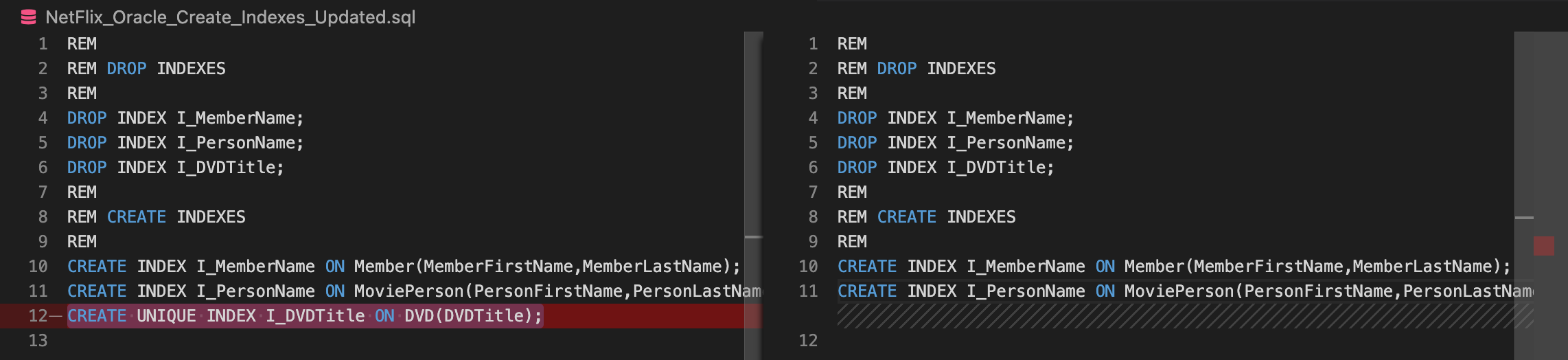
### 1. Inspect the DDL



### 2. Document DDL Changes

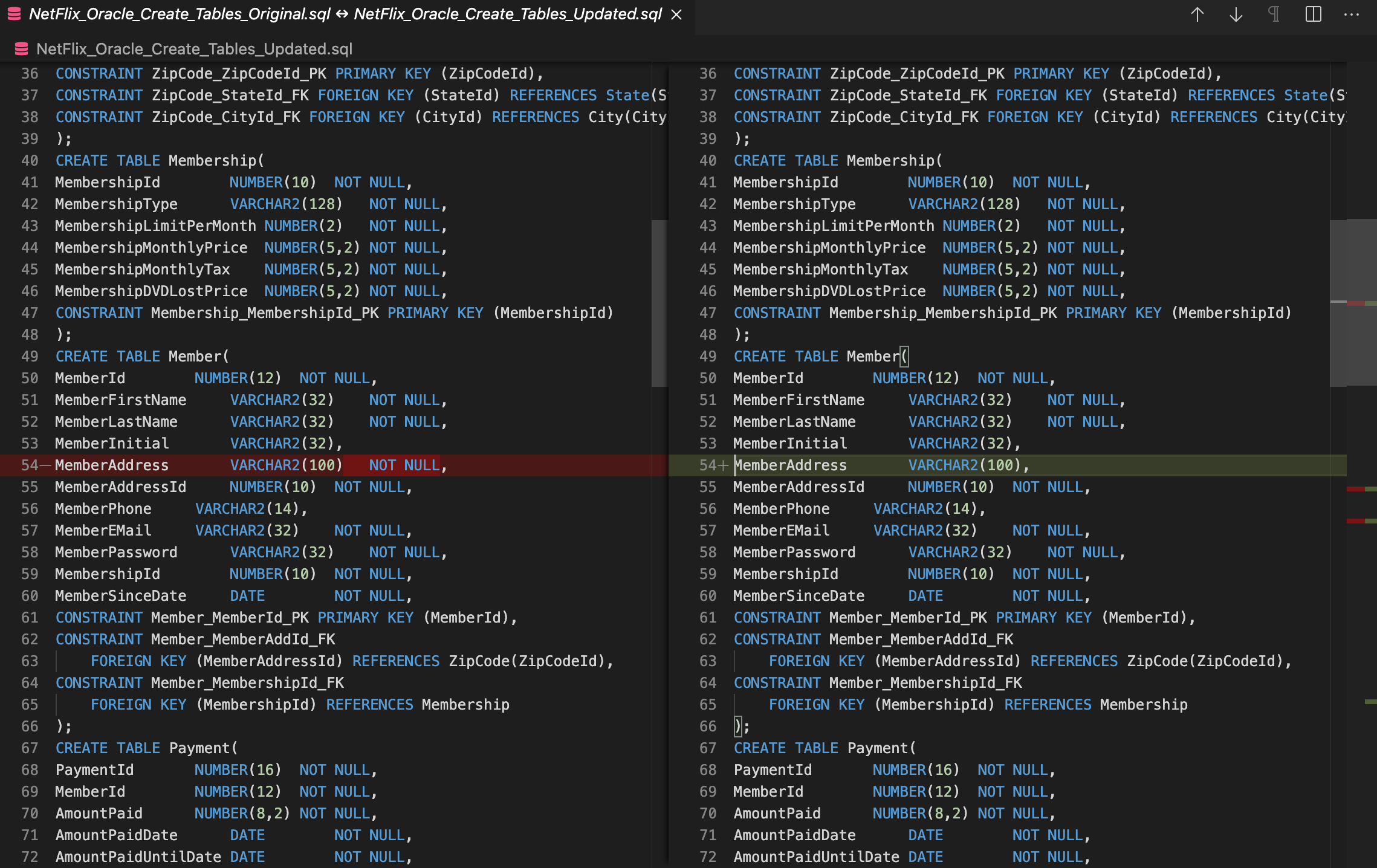
Document what DDL changes you need to make and provide the SQL and the screenshots showing the changes being made.

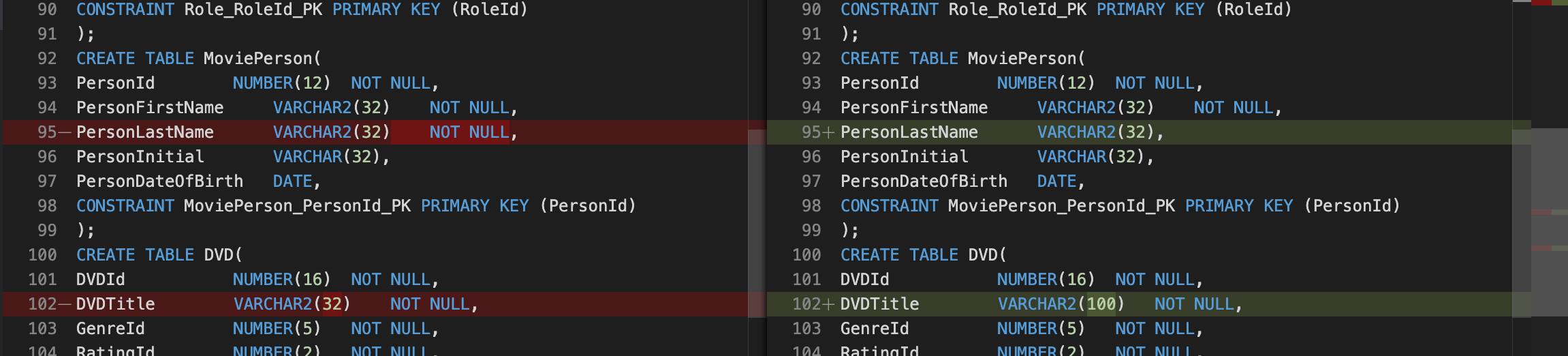
Left is Original, Right is Updated



DROP INDEX I\_DVDTitle;







ALTER TABLE member

MODIFY memberaddress VARCHAR2(100) NULL;



ALTER TABLE movieperson

MODIFY personlastname VARCHAR(32) NULL;



ALTER TABLE dvd

MODIFY dvdtitle VARCHAR2(100);



### 3. Loading the Test Data

@/Users/chrispellegrino/desktop/Netflix/dvd\_script.sql;



@/Users/chrispellegrino/desktop/Netflix/movieperson\_role\_script.sql;



@/Users/chrispellegrino/desktop/Netflix/movieperson\_script.sql;



@/Users/chrispellegrino/desktop/Netflix/dvd\_quantity\_script.sql;



@/Users/chrispellegrino/desktop/Netflix/member\_script.sql;



@/Users/chrispellegrino/desktop/Netflix/rental\_queue\_script.sql;



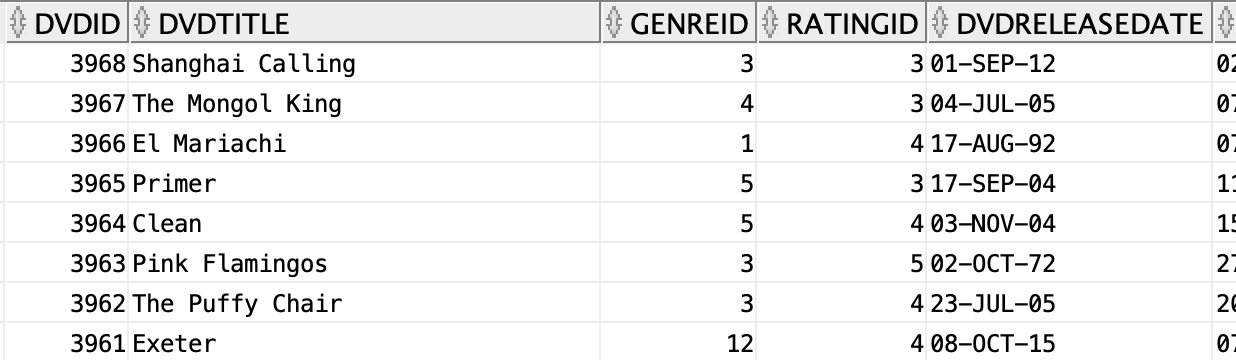
@/Users/chrispellegrino/desktop/Netflix/rental\_script.sql;



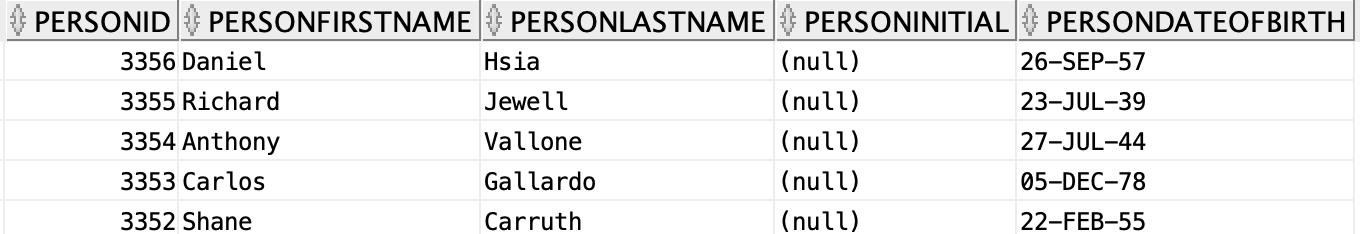
Note: The command code on the assignment (e.g. @C:\Netflix\dvd\_script.sql) was only for Windows. If you include this format with the assignment for the next time you teach this class, it would definitely save Mac user students some time.

### 4. Document Record Count

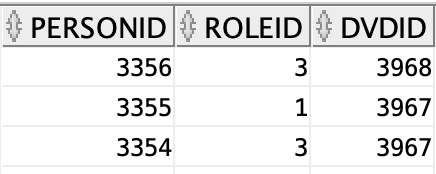
Document the record count from the loaded tables by providing the screenshot.

SELECT \* FROM dvd d ORDER BY d.dvdid DESC;

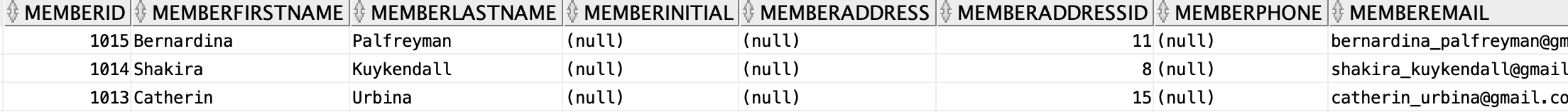
SELECT \* FROM movieperson mp ORDER BY mp.personid DESC;



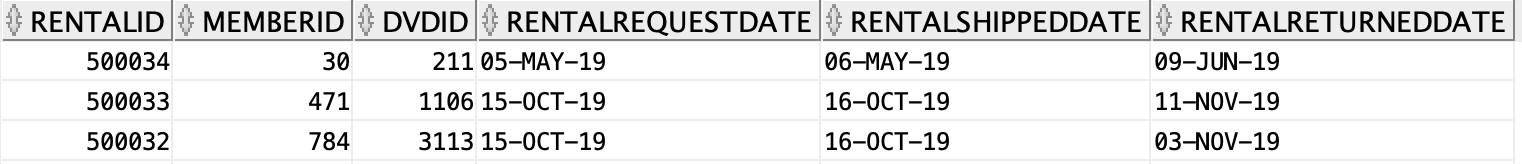
SELECT \* FROM moviepersonrole mpr ORDER BY mpr.personid DESC;



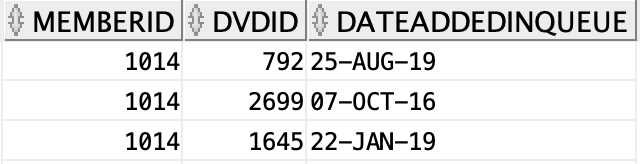
SELECT \* FROM member m ORDER BY m.memberid DESC;



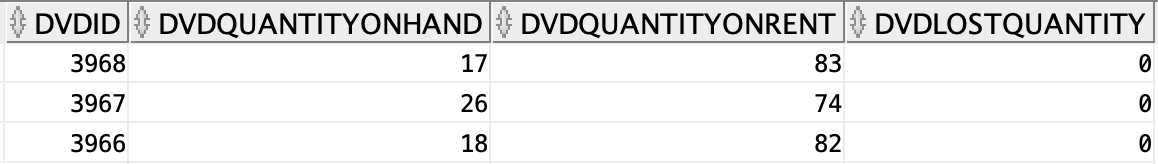
SELECT \* FROM rental r ORDER BY r.rentalid DESC;



SELECT \* FROM rentalqueue rq ORDER BY rq.memberid DESC;



SELECT d.dvdid, d.dvdquantityonhand, d.dvdquantityonrent, d.dvdlostquantity FROM dvd d ORDER BY d.dvdid DESC;



## Part B: Questions

### 1. Identify Performance and Other Scaling Problems

Suppose that the small firm for which you designed the DVD rental database has established an internet presence and their sales are growing rapidly. You have been asked to examine the database to make sure that it can grow to support at least a million transactions per day. Your assignment is to examine the DVD rental schema provided and identify performance and other scaling problems. The frequently occurring operational transactions are:

* Millions of customers updating their wish list (DVD rental queue) daily.
* The receipt of a DVD, with the corresponding mailing of the next in-stock DVD from the customer’s wish list (the rental records).
* Monthly billing of each customer’s credit card.
* The scaling issues result from customers adding data to the rental queue, the rental and the payments table, while staff frequently query from these tables for business-driven decisions. Performance degradation has been observed both from customer point of view when renting DVDs, adding Movies to the queue and making payments, and staff quiring the data.

Write a compact textual description (approximately two thirds of a page single spaced) to discuss the following topics:

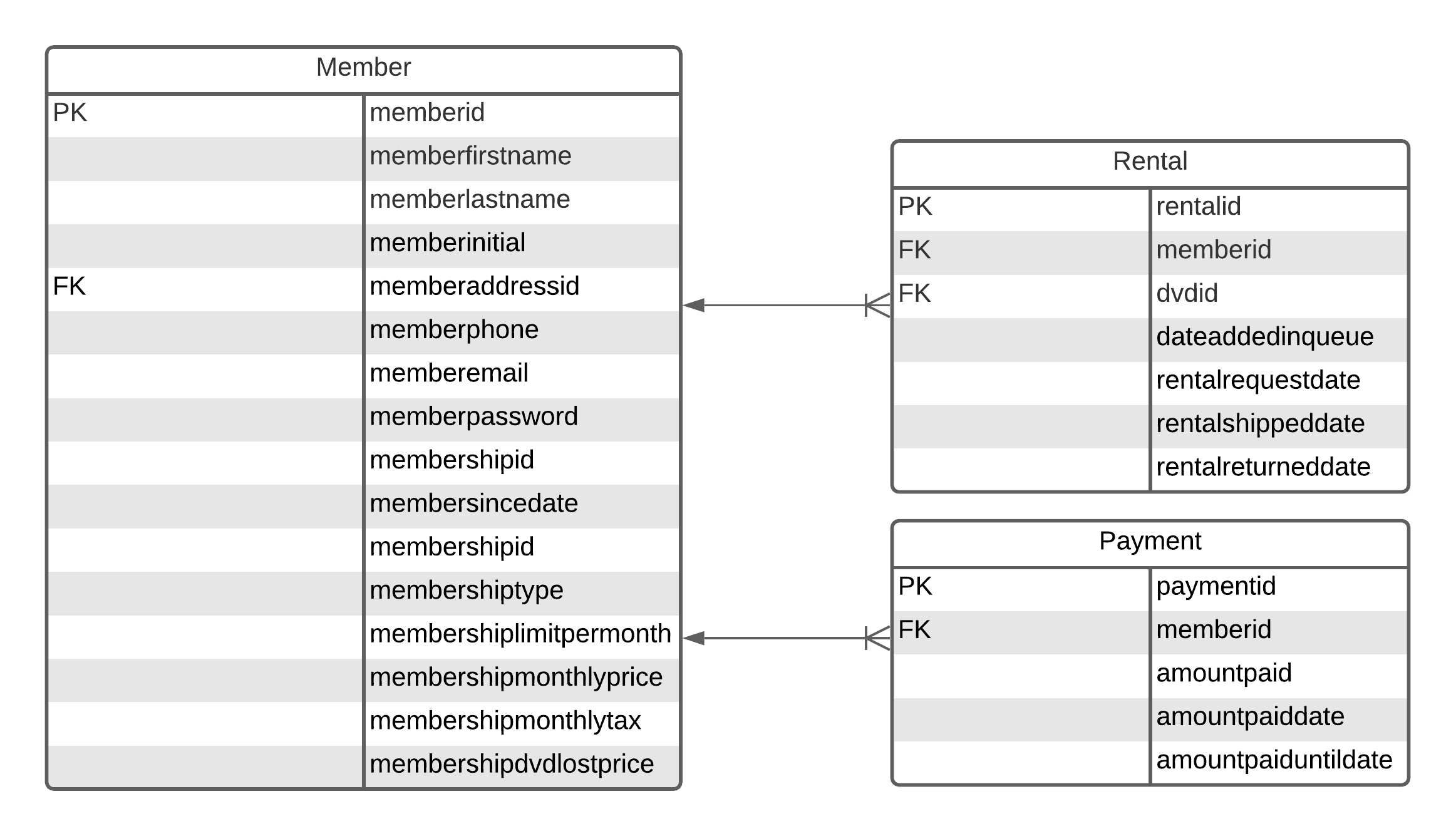
* What changes would you make to the provided schema so that it can scale to handle a million or more transactions per day? Identify the scalability-limiting schema features and propose changes that would make them scalable.
* Would you denormalize? If you would denormalize, how would you maintain the denormalizations? Justify your design changes regardless of your decision about whether to denormalize.

The scalability of the schema is limited by a bottleneck design on the member table. This design is a bottleneck because when members add DVDs to their wishlist, the Netflix database writes to the rental table, rentalqueue table, and the payments table, which all have foreign keys to the member table through the memberid primary key. The referring back to the member table costs more time for users and database developers, as the database needs to validate the information by doing this back-and-forth for every individual entry. At a minimum of one million entries, this would cause severe lag.

I would denormalize the membership table into the member table by adding its columns to the member table. As the schema is, every month, the application would have to bill the member by querying the member table for their personal information, the payment table for their payment information, the rentalqueue table for the next available dvd in their wishlist, the rental table to make sure they returned their prior dvd before sending out and charging them for the next one, and the membership table in order to get the member’s information that will populate the bill. Denormalizing the membership into the member table saves one of these steps and saves time by eliminating a JOIN, although at the cost of being able to edit the membershipid. This is an example of how sacrificing data integrity/risking data anomalies to improve performance.

Next, I would denormalize the rentalqueue table into the rental table so that there would not be an unnecessary JOIN command. This would increase speed. I would also add an index table on the rental table for valid member IDs that would be updated constantly so I could remove the the rental\_memberid\_fk constraint from having to check the member table. This would also increase speed.

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

****

### 2. Topic Discussions

A key part of growing the DVD rental business has been targeted email marketing to members. One of the suggestions to improve performance of marketing analysis has been to implement a RentalHistory table. A major concern is that as the rental history table grows, these queries may take longer and consume increasing resources. Our client is aware that as they continue to grow, they will need a data warehouse to support these marketing analyses. They do not want to undertake the effort at this time, the client does want to improve the performance of the marketing queries against the existing OLTP schema. In a later assignment you will design a dimensional data warehouse for our DVD rental business client, so we are not asking for that solution now.

Within about half a page (single spaced), **discuss the following topics**:

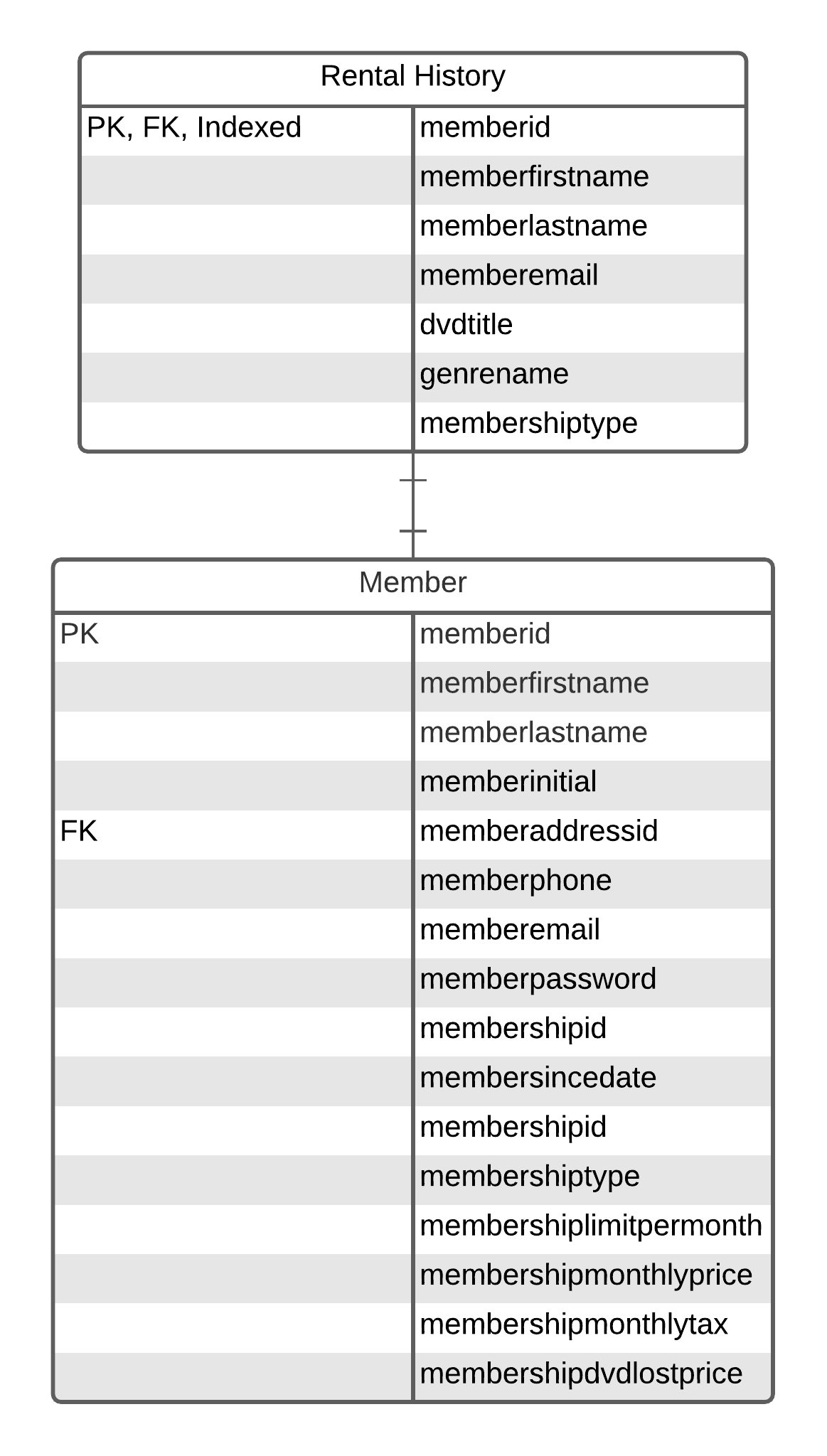
* How would you design a denormalized RentalHistory table to reduce the number of tables that need to be joined?
* How would you maintain this data, meaning how and when would it be populated and updated? Compare and contrast some different approaches (think stored procedures vs. triggers).

I would design a denormalized rental history table by including repeating data, rather than denormalizing any other tables into/with it. Even though this would take up more memory space, it would reduce the need for JOIN statements. The attributes of the table would be memberid (indexed primary key and foreign key with members table), memberfirstname, memberlastname, memberemail, dvdtitle, genrename, and membershiptype.

From these attributes, the marketing department can run queries on just this table which gives them all the information they need to contact the customer, see what titles they have watched, recommend other movies within the same genre to them, and target members with a certain type of membership for special promotions. For example, members with a membership type that allows them to rent more dvds will get more notifications. The purpose of the index on memberid is so the application does not need do a full table scan of the member table. An index on membershiptype so that members with a certain type of membership could be queried specifically, reducing querying time. The memberid would be both the primary key and the foreign key so the rental history table has a 1 to 1 relationship with the member table. One member has one rental history.

I would maintain the data by creating a stored procedure that would query the rental, membership, and the member tables and then run an UPDATE command. The update would populate the columns of the rentalhistory table to newly added data from that day. I would then use the DBMS scheduler to create a job that would schedule the procedure to execute at night, after business hours, while nobody is querying the database or editing it. This would help to maintain the data integrity as well.

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.



Explanation: The rental history table uses memberid, which is already unique, as both its primary key and foreign key, as it would have a one to one relationship with the member table. To make performance faster, an index

## Part C: Questions

For the following questions in this exercise, assume that your DVD rental client is finding that the following queries are becoming problematically slow as the database grows in size the additional sample data which you loaded will give you some context.

For each query, analyze why it may be slow, and propose ways to improve query performance either by changing the query; adding, deleting or modifying indexes, and or suggesting design changes.

You can implement the query and check its execution plan before and after your changes are made. Look to rewrite the query and explain why you made the change, then showing it in action via screenshots, as well as any additional suggestions you may have. **It is important to explain why you are proposing the change.**

Please note that these concepts are not exclusive in order to solve these questions:

* Understanding differences between joining large data sets to small data sets either through subqueries or joins
* Filtering on NULLS
* Understanding NOT IN, NOT EXISTS, DISTINCT, searching on NULLs and different data types (e.g. search on a string vs. numeric type attributes)
* Indexing concepts (different kind of indexes and where these might come into play)
* Reducing number of similar operations
* Design changes such as denormalization and or fragmentation (be careful about redundancy), and primary/foreign-key maintenance.
* Stored procedures, triggers, views, materialized/indexed views
* Use Explain Plan within the DBMS which can show the improvements and may provide hints on where the bottlenecks may be.
  + Use Explain Plan (Oracle)



### 1. Query Performance Issue 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.

SELECT DVD.DVDTitle AS "DVD Title",

SUBSTR(MoviePerson.PersonFirstName,1,10) || ' ' ||

SUBSTR(MoviePerson.PersonLastName,1,10) AS "Director Name"

FROM DVD

JOIN MoviePersonRole ON DVD.DVDId = MoviePersonRole.DVDId

JOIN Role ON MoviePersonRole.RoleId = Role.RoleId

JOIN MoviePerson ON MoviePersonRole.PersonId = MoviePerson.PersonId

WHERE

Role.RoleName = 'Director'

AND ((MoviePerson.PersonLastName = 'Spielberg')

OR

(MoviePerson.PersonLastName = 'Hitchcock' AND MoviePerson.PersonFirstName = 'Alfred'))

ORDER BY MoviePerson.PersonLastName, MoviePerson.PersonFirstName;

A) 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?

The language of the query itself is unnecessarily complicated/wordy for the data being requested. There are also unnecessary pieces of code that don’t actually help improve accuracy. The names of the directors do not need to be taken using the SUBSTR method, as their first and last names are in their own cells and thus not a part of a larger string. The role name of the director does not need to be specified, as their specific names are being queried. An easier way of searching for the directors would be by their unique personid number. Lastly, the ORDER BY clause does not need the personfirstname at the end, since it does not change the output.

B) Your recommendation to improve performance:

I propose aliases for better readability in general. I recommend simply concatenating the director’s first and last names, as they do not require a SUBSTR function to be acquired. the JOIN statements would basically be the same but with aliases for better readability. The WHERE clause should be changed completely so that the personid’s of Alfred Hitchcock and Stephen Spielberg are specified, and then only including the personlastname in the ORDER BY clause, although this in itself is unnecessary, as it only makes Alfred Hitchcock’s films appear first. The results would be the same in their overall content if the clas was removed completely.

SELECT d.dvdtitle AS "DVD Title",

mp.personfirstname || ' ' || mp.personlastname AS "Director Name"

FROM dvd d

JOIN moviepersonrole mpr ON d.dvdid = mpr.dvdid

JOIN movieperson mp ON mpr.personid = mp.personid

JOIN role r ON mpr.roleid = r.roleid

WHERE mp.personid = 2

OR mp.personid = 1617

ORDER BY mp.personlastname;



SELECT d.dvdtitle AS "DVD Title",

mp.personfirstname || ' ' || mp.personlastname AS "Director Name"

FROM dvd d

JOIN moviepersonrole mpr ON d.dvdid = mpr.dvdid

JOIN movieperson mp ON mpr.personid = mp.personid

JOIN role r ON mpr.roleid = r.roleid

WHERE mp.personid = 2

OR mp.personid = 1617;



### 2. Query Performance Issue 2

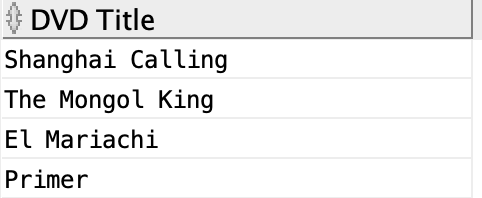
Queries like the following are run frequently to identify DVDs that a member has not rented. Note that the MemberID is passed from the application layer and may be different each time the query is run.

SELECT DVDTitle AS 'DVD Title'

FROM DVD

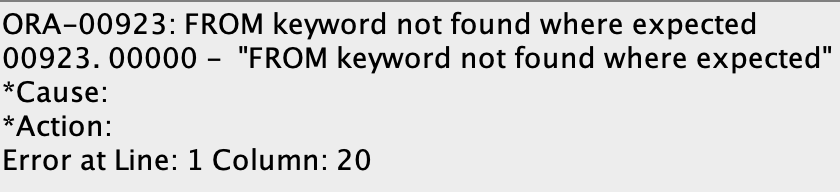
WHERE DVDId NOT IN

(SELECT DISTINCT DVDId FROM Rental WHERE MemberId = 123);



A) 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?

The alias for dvdtitle needs to be in double quotes in Oracle, otherwise it outputs an error.



The query has a subquery, which makes it take longer than if the individual simply used a JOIN statement. I wouldn’t consider the execution plan due to its need for more time to run an extra query.

B) Your recommendation to improve performance:

I would propose that the dvd title is search from the rental table an joined with the dvd table where the memberid is not equal to 123. To better show this results in the same output, I tweaked both to show the number of records after, as when compared normally they are all jumbled.

SELECT DISTINCT d.dvdtitle AS "DVD Title"

FROM rental r

JOIN dvd d ON r.dvdid = d.dvdid

WHERE memberid != 123;



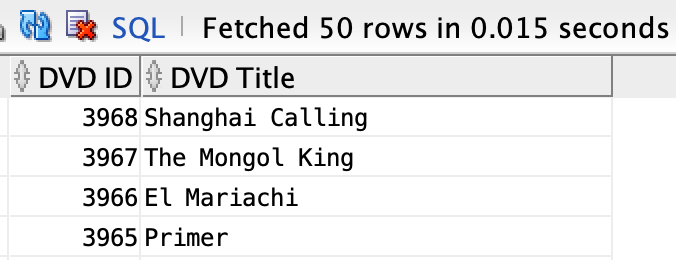
SELECT dvdid AS "DVD ID", DVDTitle AS "DVD Title"

FROM DVD

WHERE DVDId NOT IN

(SELECT DISTINCT DVDId FROM Rental WHERE MemberId = 123)

ORDER BY dvdid DESC;



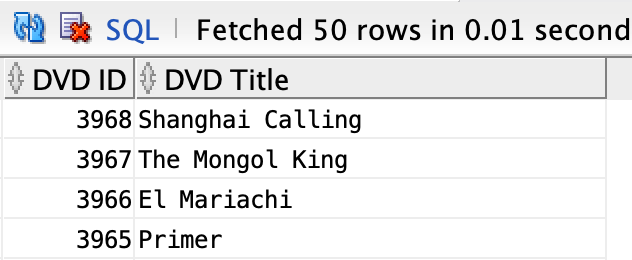
SELECT DISTINCT d.dvdid AS "DVD ID", d.dvdtitle AS "DVD Title"

FROM rental r

JOIN dvd d ON r.dvdid = d.dvdid

WHERE memberid != 123

ORDER BY d.dvdid DESC;



### 3. Query Performance Issue 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.

CREATE VIEW DVDView AS

SELECT DVDId, DVDTitle, Genre.GenreName AS Genre,

Rating.RatingName AS Rating

FROM DVD

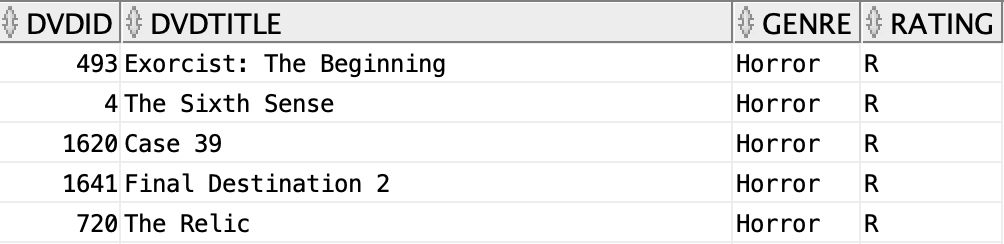
JOIN Genre ON DVD.GenreId = Genre.GenreId

JOIN Rating ON DVD.RatingId = Rating.RatingId;



Queries against this view take forms, such as:

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



A) 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?

Since the user is only looking for the movies themselves, a performance issue would be selecting \* when they could simply select dvdtitle by itself while keeping the restrictions set by the WHERE and AND clauses for genre and rating.

B) Your recommendation to improve performance:

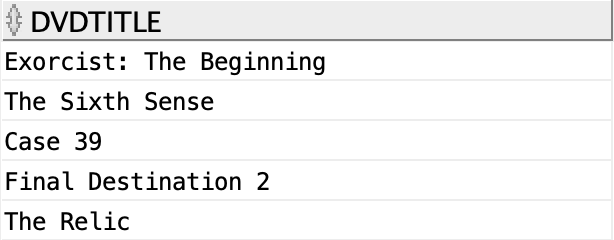
I would propose selecting only dvdtitle as opposed to all columns. This would result in less information to output while still showing the correct results.

SELECT dvdtitle

FROM dvdview

WHERE genre = 'Horror'

AND rating = 'R';



### 4. Query Performance Issue 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.

**Original code from Assignment 1:**

SELECT DISTINCT

m.memberfirstname || ' ' || m.memberlastname AS "Member Name",

d.dvdtitle AS "DVD Title",

g.genrename AS "Genre",

rtg.ratingname AS "Rating",

mp.personfirstname || ' ' || mp.personlastname AS "Director",

dc.copyid AS "DVD Copy",

r.rentalrequestdate AS "Rental Request Date",

r.rentalshippeddate AS "Rental Shipped Date",

r.rentalreturneddate AS "Rental Returned Date"

FROM member m

JOIN rental r ON m.memberid = r.memberid

JOIN dvd d ON r.dvdid = d.dvdid

JOIN rating rtg on d.ratingid = rtg.ratingid

JOIN dvd\_copy dc ON d.dvdid = dc.dvdid

JOIN genre g ON d.genreid = g.genreid

JOIN moviepersonrole mpr ON d.dvdid = mpr.dvdid

JOIN movieperson mp ON mpr.personid = mp.personid

JOIN role rl ON mpr.roleid = rl.roleid

WHERE mp.personfirstname || ' ' || mp.personlastname IN (

SELECT mp.personfirstname || ' ' || mp.personlastname

FROM movieperson mp JOIN moviepersonrole mpr ON mp.personid = mpr.personid JOIN role rl ON mpr.roleid = rl.roleid

WHERE rl.rolename = 'Director')

AND r.rentalreturneddate IS NULL;

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

There are too many JOIN commands, which make it run slower which matter more in a database with significantly more records. Seeing the members’ names and the moviepersons’ names (in this case director) having to be searched for an concatenated seems like a frequently used query, and should be made more easily accessible.

B) Your recommendation to improve performance:

I would denormalize DVD\_COPY into DVD since the only unique field is copyID to remove one of the joins. I would also create an index on the memberfirstname, memberinitial, and memberlastname from the member table, as well as another index on the personfirstname, personinitial, and personlastname from the movieperson table.

1. CREATE INDEX membername

ON member (memberfirstname, memberinitial, memberlastname)



1. CREATE INDEX moviepersonname

ON movieperson (personfirstname, personinitial, personlastname);



### 5. Query Performance Issue 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.

**Original code from Assignment 1:**

SELECT

d.dvdtitle AS "DVD Title",

g.genrename AS "Genre",

rtg.ratingname AS "Rating",

COUNT(r.dvdid) AS "Number of Rentals"

FROM dvd d

JOIN rental r ON d.dvdid = r.dvdid

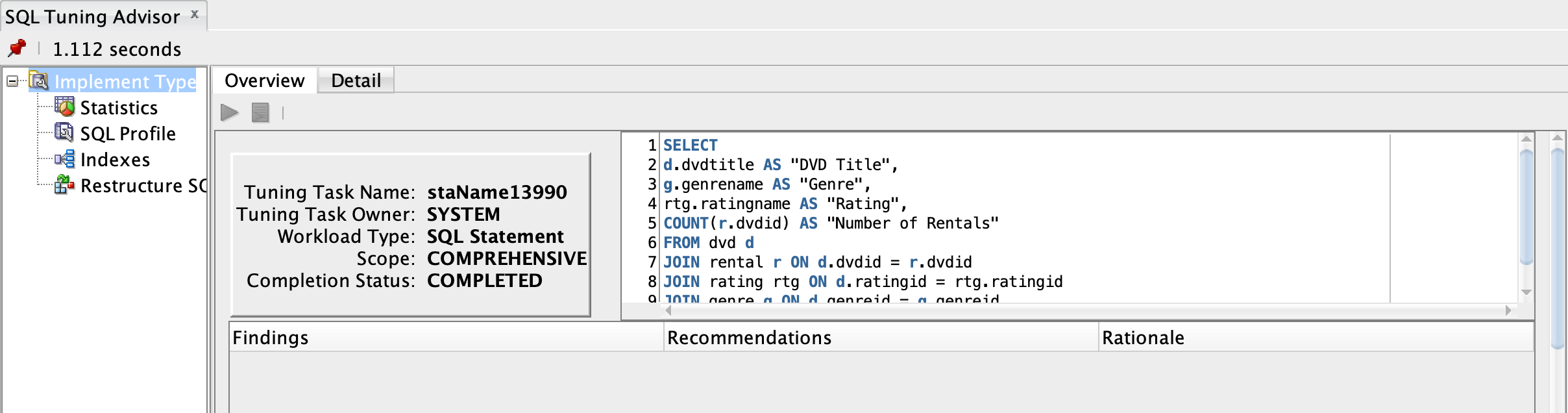
JOIN rating rtg ON d.ratingid = rtg.ratingid

JOIN genre g ON d.genreid = g.genreid

GROUP BY d.dvdtitle, g.genrename, rtg.ratingname;

A) 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?

I don’t see performance issues necessarily, but I do think this is an inefficient way of doing this common query. After using SQL Tuning Advisor, I also did not find any recommendations.



B) Your recommendation to improve performance:

I would either make the above code a stored procedure so that it can run and remain in the cache for faster access. Otherwise I would create a table with redundant data that would include dvdtitle, genrename, ratingname, and dvdid with dvdid acting as both the primary key and the foreign key. Then I would have a one to one relationship between this new table and the dvd table.

CREATE TABLE dvds\_rented\_count (

dvdid NUMBER(16) PRIMARY KEY,

dvdtitle VARCHAR2(100) NOT NULL,

genrename VARCHAR2(20) NOT NULL,

ratingname VARCHAR2(10),

CONSTRAINT FK\_DVDID\_DVD FOREIGN KEY (DVDID) REFERENCES DVD(DVDID)

);

### 6. Query Performance Issue 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.

This stored procedure is invoked when we have a lost DVD event. When given RentalId, MemberId, Amount charged (currently $25) and transaction date, this procedure will:

1. Enter a charge in the Payment table

2. Update the Rental table

3. Update the DVD table

CREATE OR REPLACE PROCEDURE Lost\_DVD -- P prefix denotes parameters

(P\_RentalId IN NUMBER, -- The rental for which DVD was lost

P\_MemberId IN NUMBER, -- The member

P\_Amount IN NUMBER, -- The amount to charge

P\_Trans\_Date IN DATE) --The transaction date and time

AS -- The V prefix denotes variables.

V\_DVDId NUMBER := 0.0;

V\_PaymentId NUMBER := 0;

V\_UntilDate DATE := NULL;

BEGIN

-- Get DVDId

      SELECT DVDId into V\_DVDId From RENTAL WHERE RentalId = P\_RentalId;

-- Get PaymentId

       SELECT COUNT(\*)+1 into V\_PaymentId FROM Payment;

-- Get AmountPaidUntilDate

SELECT AmountPaidUntilDate INTO V\_UntilDate FROM Payment

WHERE MemberId = P\_MemberId

AND PaymentId =

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

-- Now, make the changes to the database…

INSERT INTO Payment(PaymentId, MemberId, AmountPaid,

AmountPaidDate,AmountPaidUntilDate)

VALUES(V\_PaymentId,P\_MemberId,P\_Amount,P\_Trans\_Date,V\_UntilDate);

UPDATE Rental SET RentalReturnedDate = P\_Trans\_Date

WHERE RentalId = P\_RentalId;

UPDATE DVD SET DVDQuantityOnRent = DVDQuantityOnRent - 1

WHERE DVDId = V\_DVDId;

UPDATE DVD SET DVDLostQuantity = DVDLostQuantity + 1

WHERE DVDId = V\_DVDId;

END Lost\_DVD;



A) What performance issues do you see with this stored procedure:

The “Get AmountPaidUntilDate” only works if the memberID exists in the payment table, and it might not for new customers who haven’t paid yet.

The procedure has to do too many table scans that will cause it to be slow and take more time.

The UPDATE statements for DVDQuantityOnRent and DVDLostQuantity are unnecessarily separated.

B) Your recommendation to improve performance:

Add a conditional IF statement that states if a member has not rented a DVD yet then the stored procedure raises an error stating that they can’t have lost a DVD yet.

Create index on table Payment for the columns (memberid, paymentid) and a separate index for just memberid to optimize speed by avoiding a full table scan.

DVDQuantityOnRent and DVDLostQuantity could be combined into one UPDATE statement.

UPDATE DVD

SET DVDQuantityOnRent = DVDQuantityOnRent – 1,

DVDLostQuantity = DVDLostQuantity + 1

WHERE DVDId = V\_DVDId;