Mi Gao

migao@bu.edu

Assignment 3.0

Distributed Databases

MET CS 779

MET CS 779 Assignment 3.0

Table of Contents

[1. Introduction 2](#_Toc46884073)

[2. Main Goals 2](#_Toc46884074)

[3. Architecture Design 2](#_Toc46884075)

[4. Partitioning and Replication 3](#_Toc46884076)

[1 Replication 3](#_Toc46884077)

[2 Partitioning 4](#_Toc46884078)

[5. Technologies 5](#_Toc46884079)

[6. Cost 6](#_Toc46884080)

[7. Conclusion 6](#_Toc46884081)

[7. Revision History 7](#_Toc46884082)

[Bibliography 8](#_Toc46884083)

# Introduction

The fictional Netflix business will operate across North America with several regional shipping centers and headquarters for financial and billing data. There are having significant performance issues now that the scale has grown. In addition, the fictional Netflix business will now “RedBox” style locations, these are automated DVD dispensing machines which are placed in supermarkets, the fictional scenario will deploy over a thousand such DVD dispensing machines across North America so that the availability of these systems connecting to a centralized database will also become a problem.

Based on those problems, a distributed database solution has been needed to be deployed as soon as possible.

# Main Goals

Without the features already have in the original database, some performance needs to be improved or created for supporting the new demand that follows with those new facilities.

* 1. Regional shipping centers should be able to keep tracking the status of the shipment and the return
  2. Headquarters should be able to operate related financial and billing data efficiently
  3. Automated DVD dispensing machines should be above to check data from the database for making sure the information such as the stock, members’ account profile, and rental history

# Architecture Design

Based on the situation in North America, generally dividing the regions from the area of whole country is the first step of the distribution. There will have three main regions based on geographical division:

* 1. US\_WEST
  2. US\_CENTRAL
  3. US\_EAST

Dividing the whole country size by those three regions can make different shipping centers and headquarters be divided into the specific region and improve the efficiency from the first rather than the centralized database in the past. The data of particular stuffs will be processed and stored be the file management system such as DVD copy or employees in particular position of particular region.

The benefits from this upgrade of the database system are:

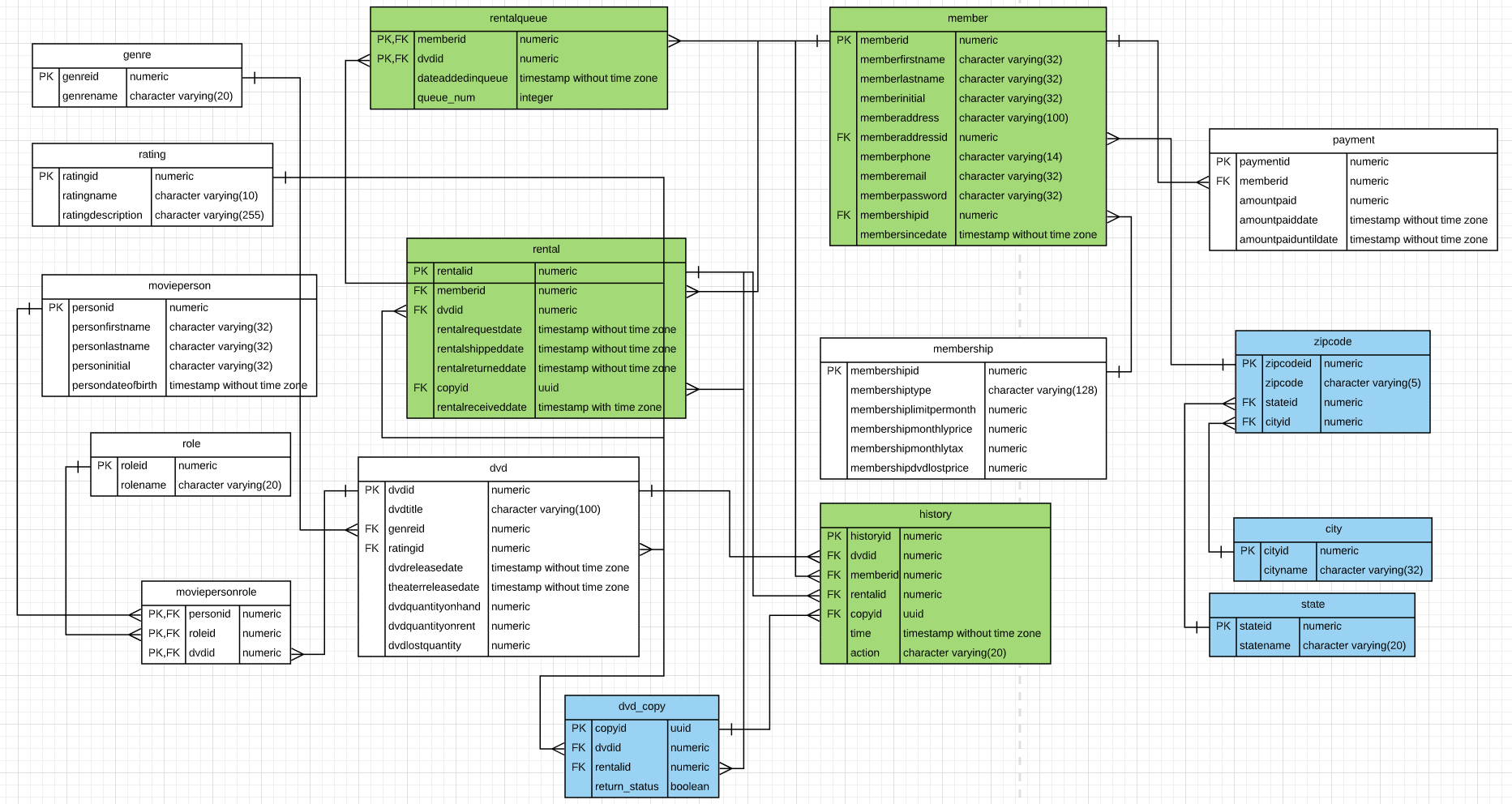
1. Easily face the expansion and growth of the business
2. Easily set up replication to maintain your data's integrity
3. High availability and salvation of failover

However, there have inevitable disadvantages:

1. Difficult to maintain
2. Difficult to set up

# Partitioning and Replication

## Replication



Get start from this logical ERD. There have three different tables have been set with different colors.

First of all, the tables with white color have the highest priority to be replicate to any position in the new database based on their unique properties. No matter which one, it has the characteristic which is rare variation; from the moment it is created and inserted, it is rarely changed. They are needed and connected with almost important transactions because each transaction can not leave with three main things: the member, DVD, and the dealing. Thus, they have to be survived from failures so that no matter where the database fails happen, the data still can be obtained from another places.

The next one has slightly higher distinguishability with green color. For example of history table, generally it will be check by regional shipping centers and local headquarters for overseeing of each actions happening in the process. Also, the member table could depend on where the customer lives. Otherwise, they could be replicate in all the instance if is needed. For example, it can be full replicated if the role considers the member may be traveling around.

Then, the most suitable to be fragmented are the blue-marked tables. Whether dvd\_copy or cities or states, they are scattered around the places and coordinated by local agencies or file management systems. For example, regional shipping center can choose the closest inventory center to ship the copy to the customer. Also, automated DVD dispensing machines in different place can automatically calculate the shortest arrival time or other features based on connecting and checking to the local database center.

## Partitioning

When the write operation increases, then the master-slave database will spend more time on data synchronization, this time the server is also overwhelmed. The most common use of vertical partitioning is to reduce the I/O and performance costs associated with frequently accessed items. The example such as the dummy tables of the data show.

Table 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Rentalid | Dvdid | Memberid | copyid | Requestdate | Shippeddata | Receiveddata | Returndate |
| 1 | 4 | 10 | 77DR51 | 02-22-2019 | 07-22-2019 | NULL | NULL |
| 2 | 11 | 5 | 30ACT2 | 04-21-2020 | 04-22-2020 | 04-24-2020 | 05-01-2020 |
| 3 | 7 | 30 | 1TRB23 | 07-10-2020 | NULL | NULL | NULL |

Table 2

|  |  |  |  |
| --- | --- | --- | --- |
| Rentalid | Dvdid | Memberid | copyid |
| 1 | 4 | 10 | 77DR51 |
| 2 | 11 | 5 | 30ACT2 |
| 3 | 7 | 30 | 1TRB23 |

Table 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requestdate | Shippeddata | Receiveddata | Returndate | Requestdate |
| 02-22-2019 | 07-22-2019 | NULL | NULL | 02-22-2019 |
| 04-21-2020 | 04-22-2020 | 04-24-2020 | 05-01-2020 | 04-21-2020 |
| 07-10-2020 | NULL | NULL | NULL | 07-10-2020 |

In this example, when the application displays product details in table 1 to the regional shipping center. The different attributes of the item are stored in different partitions. One partition store frequently accessed data, and the other partition stores inventory data. After partitioning, it routinely queries the rental ID, DVD ID, members’ ID, and DVD copies’ ID in table 2. The date with related actions is kept in separate partitions in table 3, because these are usually used and updated together or more frequently. If these two tables and other tables have a join table query, then you can only split the original SQL statement, query one table first, and then query another. Although this will consume more performance than the large amount of data synchronization, the burden is reduced a lot.

Horizontal partitioning is also meaningful for particular values of the attributes. For example, in the full version of dvd\_copy table with full information records, the controller in warehouse may need to query like this to search the items in the stock:

*Select dvd.lendable, dvd\_copy.state, dvd\_copy.location from dvd inner join dvd\_copy on dvd.dvdid = dvd\_copy.dvdid where dvd\_copy.location = 'boston';*

*Select dvd.lendable, dvd\_copy.state, dvd\_copy.location from dvd inner join dvd\_copy on dvd.dvdid = dvd\_copy.dvdid where dvd\_copy.location = 'los angeles';*

Therefore, setting up the distributed database to speed up queries execution in different position become more and more meaningful. Based on this idea, the intersection of the table partitions should be null, and the union of table partitions should contain all rows of the initial table.

Furthermore, the example of the business is too hot: the number of visits has suddenly increased from 1 million to 10 million per day. At this time, data can be separated, and we can allocate according to the ID of the member. For example, in the form of %2 or %10, of course, this form has a great limitation on future expansion. When I increase from 10 partitions to 20, all data must be repartitioned, then Will be a very large amount of calculation; the following provides several common algorithms:

1. Hash algorithm: the method of using memberid%
2. Range: it can be divided according to the memberid character value range, such as 1-1000 is one area, 1001-2000 is another area, etc.

# Technologies

Without those products from the companies almost people already knew, CockroachDB comes into my view uniquely. CockroachDB is a distributed SQL database built on a transactional and strongly-consistent key-value store. It scales horizontally; survives disk, machine, rack, and even datacenter failures with minimal latency disruption and no manual intervention; supports strongly-consistent ACID transactions; and provides a familiar SQL API for structuring, manipulating, and querying data (Cockroach Labs, 2020).

CockroachDB is ideal for applications that require accurate, usable and correct data and millisecond response time (regardless of size). It is designed to automatically replicate, rebalance and restore with minimal configuration and overhead operations. The cases include:

* Distributed or replicated OLTP
* Multi-datacenter deployments
* Multi-region deployments
* Cloud migrations
* Infrastructure initiatives built for the cloud

CockroachDB returns a single-line read in 2ms or less, and returns a single-line read in 4ms or less. It supports various SQL and operation adjustment practices to improve query efficiency.

Partitioning in CockroachDB can be also implement successfully. For example, we have the member table. Now there has a requirement. For some reasons, the manager wants to divide the member information into three parts and store them in different systems. For Figure.1,

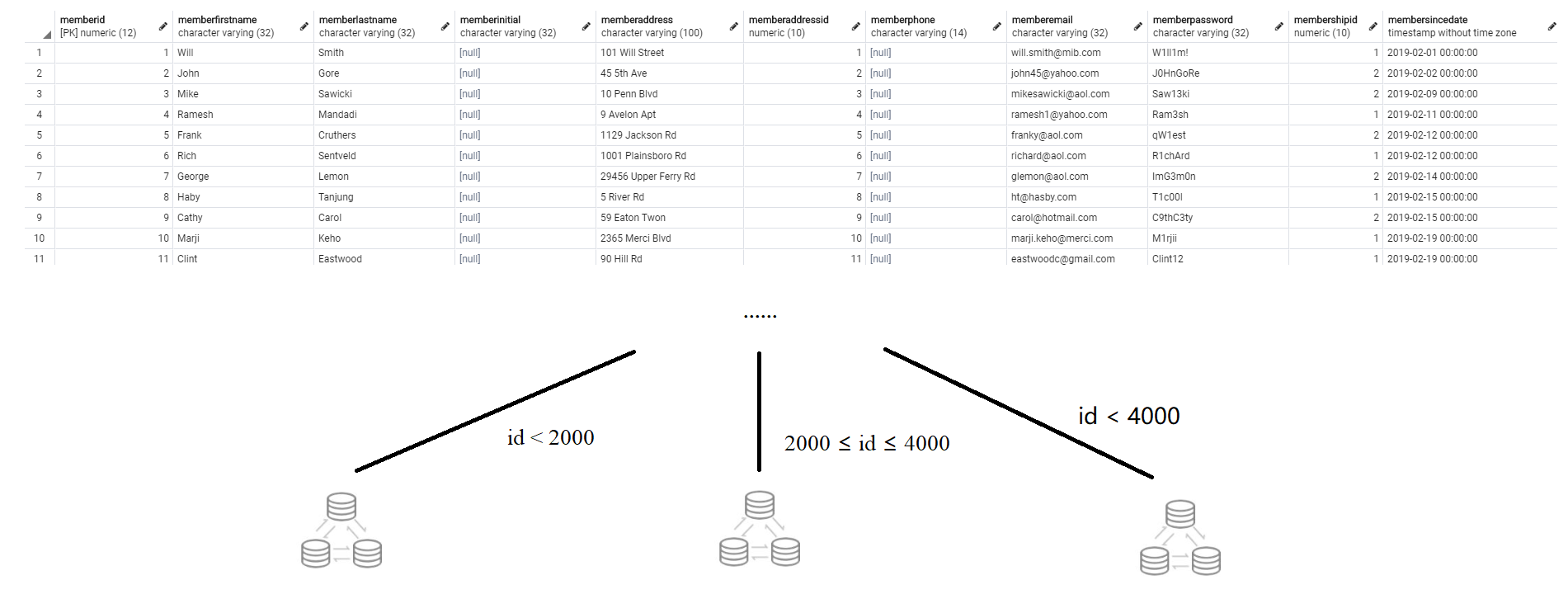


Figure 1

CockroachDB is a distributed database, and data is stored in a range of data blocks as a hash. Therefore, a cluster of 9 nodes can be grouped with 3 nodes each, and called 1, 2, 3 clusters from top to bottom. Use the patitioning function to distribute those smaller than 2000 into 1 group, those larger than 2000 and smaller than 4000 are placed in the second group, and those larger than 4000 are placed in the third cluster. In this way, for the business layer, the table is still a table, but the data has been stored separately as required, which virtually brings great convenience to the business layer.

Moreover, because CockroachDB supports the PostgreSQL wire protocol, it is easy to make the application communicate with Cockroach; just find and start creating a PostgreSQL language-specific driver. Because of the implementations of the assignment I did are all using PostgreSQL, this becomes an attractive point.

# Cost

Because of CockroachDB product layering strategy, while providing free community edition and paid enterprise edition (“Cockroach and the Source Available Future”, 2019). The software cost depends on the business role which is rental or service; it can be free or chargeable. Then, the extra cost will focus on the regional computer room or server deployment; they depend on the density of the total inputs the enterprise wants.

# Conclusion

This is one way from many plans of distributed database solution based on CockroachDB and PostgreSQL which I was most familiar with. And the details about the actual implementation will be much more difficult and complex than this article; and also have more things need to be considered. However, I give the general reference suggestions to solve current problems they are facing in regional shipping center, headquarters, “Redbox,” and increasing customers.

# 7. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Version** | **Description** |
| Mi Gao | 07/28/20 | 1.0 | Initial Document Creation |
| Mi Gao | 07/29/20 | 1.9 | Added more headers to provide better example and diagram |

# Bibliography

Cockroach Labs (2020). Frequently Asked Questions. Retrieved from https://www.cockroachlabs.com/docs/stable/frequently-asked-questions.html

O'Grady, S. (2019). Cockroach and the Source Available Future. Retrieved from https://redmonk.com/sogrady/2019/06/21/cockroach-source-available/