Project Report

Overall Status

The project is completely implemented to our knowledge.

Implementation:

For the map-reduce portion of the project, we simply copied the WordCount code and replaced the map function. The reduce function was able to remain unchanged because we used the year range and genres as the key and simply summed the instances of each as was done for words in WordCount. The map function takes the lines given to it, splits them into individual lines, then for each line, we split it into a list of the characteristics. We check a few disqualifiers like ensuring it is a movie (or tvMovie), that the rating is >= 7.5, and whether it is in the proper greater year range (1991-2020). If all of those tests pass, we create the year range portion of the key. Finally, we check if the genre list contains each pair of genres and add a row to the output with those genres as part of the key. The data types of the input, key, value, and output are all the same as they were in WordCount.

Analysis Results

Task 1 – M/R

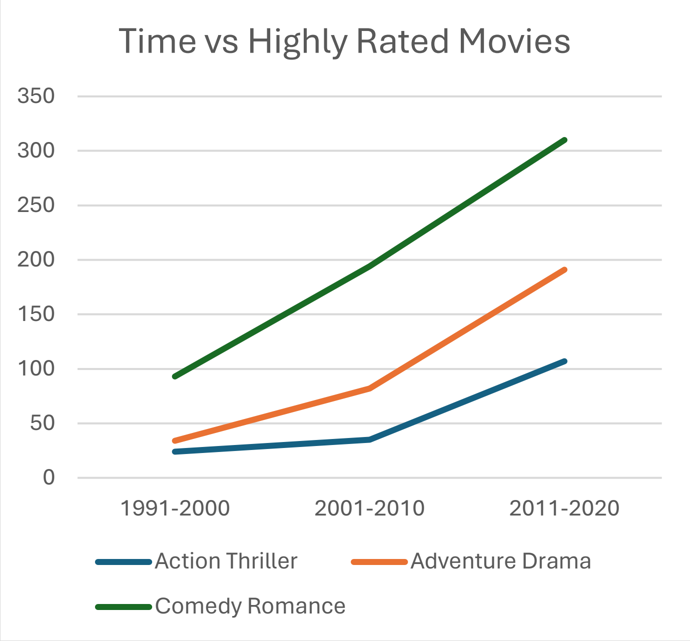
MapReduce.java Output:

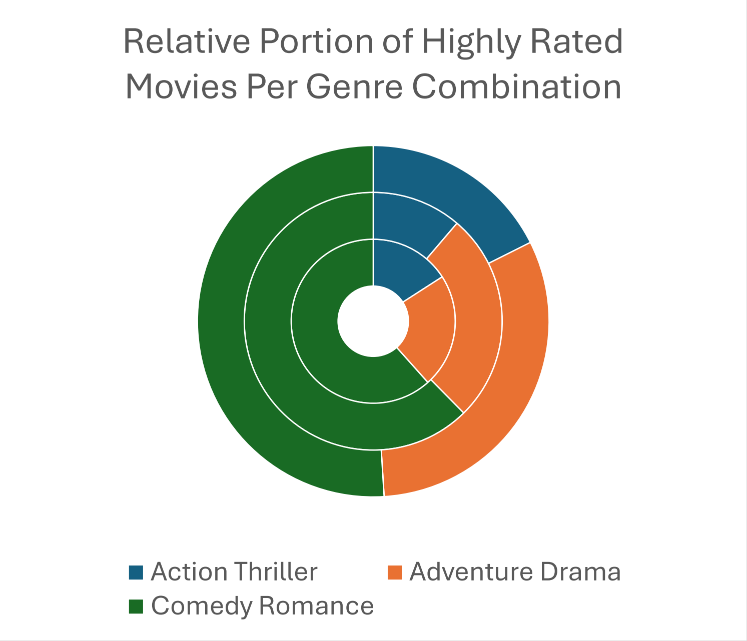
```
[1991-2000],Action;Thriller     24
[1991-2000],Adventure;Drama     34
[1991-2000],Comedy;Romance      93
[2001-2010],Action;Thriller     35
[2001-2010],Adventure;Drama     82
[2001-2010],Comedy;Romance      194
[2011-2020],Action;Thriller     107
[2011-2020],Adventure;Drama     191
[2011-2020],Comedy;Romance      310
```

Above is the output of MapReduce.java. The results show the 10-year period, the genre combination, and the count of movies with a rating above 7.5.

Overall, there were more resulting movies during the newer 10-year periods. This can be seen in the first graph where the number of occurrences increase over time.

Additionally, there is a relative increase in the number of occurrences in the adventure/drama genre combination over time. This can be seen in the second graph where the adventure/drama portion increases in size over time while the other genres either stay roughly the same or decrease in relative size





Task 2 – SQL

```
-- Get the table with the Title, Rating, Release Year and Genre List of the top 5 rated movies with at least
-- 150,000 votes, of the genres Adventure and Drama at least, and from the time period from 1991 to 2000.
Select PRIMARYTITLE, AVERAGERATING, STARTYEAR, GENRES
FROM imdb00.TITLE_BASICS T join imdb00.TITLE_RATINGS R on T.TCONST = R.TCONST
WHERE R.NUMVOTES >= 150000
AND T.GENRES LIKE '%Adventure%Drama%'
AND T.TITLETYPE LIKE '%ovie'
AND T.STARTYEAR BETWEEN 1991 AND 2000
ORDER BY R.AVERAGERATING DESC
FETCH FIRST 5 ROWS ONLY;

-- Output:

-- PRIMARYTITLE                   | AVERAGERATING | STARTYEAR | GENRES
-- ---------------------------------------------------------------------------------------
-- Gladiator                      | 8.5           | 2000      | Action,Adventure,Drama
-- The Lion King                  | 8.5           | 1994      | Adventure,Animation,Drama
-- Almost Famous                  | 7.9           | 2000      | Adventure,Comedy,Drama
-- Crouching Tiger, Hidden Dragon | 7.9           | 2000      | Action,Adventure,Drama
-- Cast Away                      | 7.8           | 2000      | Adventure,Drama,Romance


-- Store the explaination of the plan for the given query in a known table.
EXPLAIN PLAN FOR (
SELECT PRIMARYTITLE, AVERAGERATING, STARTYEAR, GENRES
FROM imdb00.TITLE_BASICS T JOIN imdb00.TITLE_RATINGS R ON T.TCONST = R.TCONST
WHERE R.NUMVOTES >= 150000
AND T.GENRES LIKE '%Adventure%Drama%'
AND T.TITLETYPE LIKE '%ovie'
AND T.STARTYEAR BETWEEN 1991 AND 2000
ORDER BY R.AVERAGERATING DESC
FETCH FIRST 5 ROWS ONLY
);

-- Output:

-- Explained.


-- Get the contents of the table populated by the EXPLAIN query.
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

-- Output:

-- Plan hash value: 2653010624
-- ------------------------------------------------------------------------------------------------
-- | Id  | Operation                       | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-- ------------------------------------------------------------------------------------------------
-- |   0 | SELECT STATEMENT                |               |     5 | 10250 |    3846 (1)| 00:00:01 |
-- |*  1 |  VIEW                           |               |     5 | 10250 |    3846 (1)| 00:00:01 |
-- |*  2 |   WINDOW SORT PUSHED RANK       |               |    57 |  6555 |    3846 (1)| 00:00:01 |
-- |   3 |    NESTED LOOPS                 |               |    57 |  6555 |    3845 (1)| 00:00:01 |
-- |   4 |     NESTED LOOPS                |               |  1380 |  6555 |    3845 (1)| 00:00:01 |
-- |*  5 |      TABLE ACCESS FULL          | TITLE_RATINGS |  1380 | 23460 |    1084 (2)| 00:00:01 |
-- |*  6 |      INDEX UNIQUE SCAN          | SYS_C00547784 |     1 |       |       1 (0)| 00:00:01 |
-- |*  7 |     TABLE ACCESS BY INDEX ROWID| TITLE_BASICS  |     1 |    98 |       2 (0)| 00:00:01 |
-- ------------------------------------------------------------------------------------------------
-- Predicate Information (identified by operation id):
--
-- ---------------------------------------------------
--    1 - filter("from$_subquery$_004"."rowlimit_$$_rownumber"<=5)
--    2 - filter(ROW_NUMBER() OVER ( ORDER BY INTERNAL_FUNCTION("R"."AVERAGERATING") DESC )<=5)
--    5 - filter("R"."NUMVOTES">=150000)
--    6 - access("T"."TCONST"="R"."TCONST")
--    7 - filter("T"."GENRES" LIKE U'%Adventure%Drama%' AND "T"."TITLETYPE" LIKE U'%ovie'
--               AND TO_NUMBER("T"."STARTYEAR")>=1991 AND TO_NUMBER("T"."STARTYEAR")<=2000 AND
--               "T"."GENRES" IS NOT NULL AND "T"."TITLETYPE" IS NOT NULL)
```

Query Plan Analysis

From the information above we can determine the order and how the DBMS retrieves the final result. Id 1 and 2 are related to the final result and don't have much impact until the bulk of the query is executed. We can see that the DBMS first iterates over all rows of TITLE_RATINGS and keeps the rows with NUMVOTES>=150000. This is the nested for loop. These results are then matched with the corresponding rows in TITLE_BASICS through the use of an index on ROWID (TCONST). Finally, these now joined tables must be filtered based on the other conditionals such as genres and titletype and staryear. This is the first for loop. The result of the bulk of the query are passed to Id 1 and 2 as mentioned earlier and these limits the results along with order them.

From our analysis, we can tell that this query is executed in an optimal fashion. It first filters based on NUMVOTES and then joins the tables instead of the other way around. This is done because the DBMS doesn't want to spend time and effort to join a row if it can be excluded with the information already available in that row. We can generally tell that there is not an index on some of the conditionals we tested for such as NUMVOTES or TITLETYPE, because if there were indexes, the DBMS would probably have used them. There are most likely no indexes on these values due to these kinds of queries being less common.

File Descriptions

Submitted files.

- 4331-5331_Proj3Sprint24_team_t.sql
  - Contains the English queries, SQL queries and their outputs.
- MapReduce.java
  - Contains the source code for the M/R.
- MR-output.txt
  - The result of the MapReduce function.
- StatsProj3.xlsx
  - The Excel spreadsheet we used to create the above plots.
- Project_report.pdf
  - This file.

Division of Labor

Landon Moon

- 3/21/2024 – 2.5 Hours – Hadoop setup
- 3/28/2024 – 2.5 Hours – Script setup for Hadoop
- 4/04/2024 – 1 Hour – Report writing/formatting

Jacob Holz

- 3/21/2024 – 2.5 Hours – Hadoop setup
- 3/28/2024 – 2.5 Hours – Script setup for Hadoop
- 4/01/2024 – 2 Hours – Project setup and first test
- 4/04/2024 – 1 Hour – Report writing/formatting