DataBase - 4
Q1
part a
a.1.

- Calculate the size of a row in the Movies table movieId: 8 bytes title: 10 bytes rating: 8 bytes year: 8 bytes duration: 8 bytes genre: 10 bytes Total size of one row in Movies = 8 + 10 + 8 + 8 + 8 + 10 = 52 bytes
- Calculate the number of rows per block Block size = 1,000 bytes Row size = 52 bytes Rows per block = Block size / Row size = 1,000 / 52 ≈ 19 rows per block
- Calculate the number of blocks needed for the Movies table Total rows in Movies = 10,000 Rows per block = 19 Total blocks needed = Total rows / Rows per block = 10,000 / 19 ≈ 526.32 ≈ 527 blocks
- Calculate the cost of reading these rows Since there are no indexes, a full table scan is required The entire table must be scanned, resulting in the full number of blocks being read Cost of reading the full table = Total blocks needed = 527 blocks
- The cost of computing the query without indexes, which involves a full table scan of the Movies table, is: 527 blocks

a.2.

- We will use the following formula to calculate the optimal branching factor: $\left\lfloor \frac{b+s}{p+s} \right\rfloor$, Given: Block size (b): 1,000 bytes, Value size (s): 8 bytes (duration), Pointer size (p): 8 bytes
- $\left\lfloor \frac{1000+8}{8+8} \right\rfloor = 63$ Therefore, the optimal branching factor of the index is 63.

a.3.

- To calculate the cost, we need to determine the height of the B+ tree. The height is determined by the logarithm of the number of entries divided by the branching factor.
- Height of the tree: $\log^{n}_{\lceil \frac{d}{2} \rceil} = \log^{10000}_{\lceil \frac{63}{2} \rceil} \approx 3$
- The values in the duration field in the Movies table are uniformly distributed in the range [1, 200]. Given the query condition 'WHERE duration > 100', we need to calculate the fraction of rows that satisfy this condition.
- The fraction of rows that meet the condition can be calculated as: $\frac{200-101+1}{200-1+1} = \frac{100}{200} = \frac{1}{2}$

- From the total number of rows, the number of matching rows is: $1000 * \frac{1}{2} = 5000$
- Number of matching table rows (m): 5,000, Branching factor (d): 63
- Leaves containing matching values (at most): $\left\lceil \frac{m}{\lceil d/2 \rceil - 1} \right\rceil = \left\lceil \frac{5000}{\lceil 63/2 \rceil - 1} \right\rceil \approx 162$
- Therefore, the total cost is: 3 (height of the tree)+162 (leaf node accesses)=165
- The cost of computing the query using the index is 165 blocks.

b.1.

- We will use the following formula to calculate the optimal branching factor: $\left\lfloor \frac{b+s}{p+s} \right\rfloor$, Given: Block size (b): 1,000 bytes ,Value size (s): 10 bytes (genre), Pointer size (p): 8 bytes
- $\left\lfloor \frac{1000+10}{8+10} \right\rfloor = 56$ Therefore, the optimal branching factor of the index is 56.

b.2.

- To calculate the cost, we need to determine the height of the B+ tree. The height is determined by the logarithm of the number of entries divided by the branching factor.
- Height of the tree: $\log^{n}_{\lceil \frac{d}{2} \rceil} = \log^{10000}_{\lceil \frac{56}{2} \rceil} \approx 3$
- Given that there are 4 categories for the genre, we divide the total number of rows by 4:
- $\frac{10{,}000}{4} = 2,500$ rows
- Each leaf node can contain at least: $\lceil \frac{d}{2} \rceil - 1 = \lceil \frac{56}{2} \rceil - 1 = 28 - 1 = 27$
- To find the 2,500 rows in the leaves: $\lceil \frac{2500}{27} \rceil \approx 93$
- Thus, the total cost to access the rows in the leaf nodes: 3 (height of the tree)+93 (leaf node accesses)+527 (data block accesses)=623
- The cost of computing the query using the index is 623 blocks.

c.1.

- We will use the following formula to calculate the optimal branching factor: $\left\lfloor \frac{b+s}{p+s} \right\rfloor$, Given: Block size (b): 1,000 bytes ,Value size (s): 10 bytes (genre), Value size (s): 8 bytes (duration), Pointer size (p): 8 bytes
- $\left\lfloor \frac{1000+(10+8)}{8+(10+8)} \right\rfloor = 39$ Therefore, the optimal branching factor of the index is 39.

- To calculate the cost, we need to determine the height of the B+ tree. The height is determined by the logarithm of the number of entries divided by the branching factor.
- Height of the tree: $\log^{n}_{\lceil \frac{d}{2} \rceil} = \log^{10000}_{\lceil \frac{39}{2} \rceil} \approx 4$
- Each leaf node can contain at least: $\lceil \frac{d}{2} \rceil - 1 = \lceil \frac{39}{2} \rceil - 1 = 20 - 1 = 19$
- The number of matching rows is still 2500. Thus, we expect to find the 2500 matching rows in the leaf nodes. $\lceil \frac{2500}{19} \rceil \approx 132$
- Finally, we calculate the cost of accessing the matching rows. The total cost is at most: 4+132=136

part B

a. To calculate the cost of the Block Nested Loops Join, we use the following formula:

For each M−2 blocks of the outer relation S :Read inner relation R, block by block, write output to the output block, and flush when full

The total cost is calculated as:

- Read outer relation once: $B(S)$
- Read inner relation $\frac{B(S)}{M-2}$ times: $B(R)$
- Thus, the cost formula is: $B(S) + B(R) \left\lceil \frac{B(S)}{M-2} \right\rceil$
- Movies Table:
  - Each row size: 8+8+8+8+10+10=52 bytes
  - Number of rows per block: $\lfloor \frac{1000}{52} \rfloor \approx 19$
  - Number of blocks for Movies: $\lceil \frac{10000}{19} \rceil \approx 527$
- PlaysIn Table:
  - Each row size: 8+8+10=26 bytes
  - Number of rows per block: $\lfloor \frac{1000}{26} \rfloor \approx 38$
  - Number of blocks for Movies: $\lceil \frac{100000}{38} \rceil \approx 2632$
- For Movies as the outer relation and PlaysIn as the inner relation:
  - $527 + 2632 \left\lceil \frac{527}{52-2} \right\rceil = 527 + 28947 = 29474$

b.2.

- The minimum buffer size is:
  - $B(S) + B(R) \left\lceil \frac{B(S)}{M-2} \right\rceil \Longmapsto 527 + 28947 \left\lceil \frac{527}{M-2} \right\rceil \geqslant 29474 \Longmapsto M = 50$
- The minimum buffer size 50

Question 2, Part B:

Given the expression: $\pi_{A,D}\sigma_{B=20\bigwedge D<5}(R(A,B)\bowtie S(A,C,D))\pi A, D\sigma B = 20\bigwedge D<5(R(A,B)\bowtie S(A,C,D))$, where:

The projection does not eliminate duplicates.

Block size is 10 bytes.

Attribute sizes are: B(R)=4,000B(R)=4,000, B(S)=1,200B(S)=1,200.

Block sizes are 2,000 bytes.

Relation R has an index on attribute B with negligible access cost.

V(R,B)=200, V(S,A)=1000.

A is a key in relation R, and R has 70 buffer blocks.

1. Number of rows in the result?

   Let's calculate the number of rows (t(S)t for S and t(R) for R) based on the given information:

   Calculating t(S): Block size for S: 2000 bytes

   - Row size for S: 10*3=30 bytes (each row in S)
   - Rows per block in S:
   - 2000 bytes per block : 30 bytes per row S0
   - $\lfloor\frac{2000}{30}\rfloor$=66

   Therefore, t(S) (number of rows in S): t(S)=B(S)×Rows per block in S=1200×66 = 79,200 So, t(S)=79,200.

   Calculating t(R): Block size for R: 2000 bytes

   - Row size for R: 10*2=20 bytes (each row in R)
   - Rows per block in R:
   - 2000 bytes per block : 20 bytes per row SO
   - $\lfloor\frac{2000}{20}\rfloor$=100

   Therefore, t(R) (number of rows in R): t(R)=B(R)×Rows per block in R=4000×100=400,000 So, t(R)=400,000.

   Summary of calculations:

   - t(S)= 79,200(number of rows in S)
   - t(R)=400,000(number of rows in R)

   Here is the SQL query based on the Given expression :

   SELECT S.D,S.A

   FROM S,R

   WHERE R.B = 20 AND S.D < 5 AND S.A=R.A;

we have learned that to calculate the number of rows in result based on this query we should use this:
$\frac{T(S)*T(R)}{3*max(v(R,A),v(S,A))*V(R,B)}$

- Range of D is unknown so Number of rows in result: $\frac{T(S)}{3}$
- Number of rows in result assuming uniform distribution of B so Number of rows in result : $\frac{T(R)}{V(R,B)}$
- when S.A = R.A we learned that we should dived the result on max(v(R,A),v(S,A))
- because S(A) is key so V(R,A) = T(R)

SO $\frac{T(S)*T(R)}{3*max(v(R,A),v(S,A))*V(R,B)} = \frac{79,200*400,000}{3*max(400,000,1000)*200}$ =132 Number of rows in the result

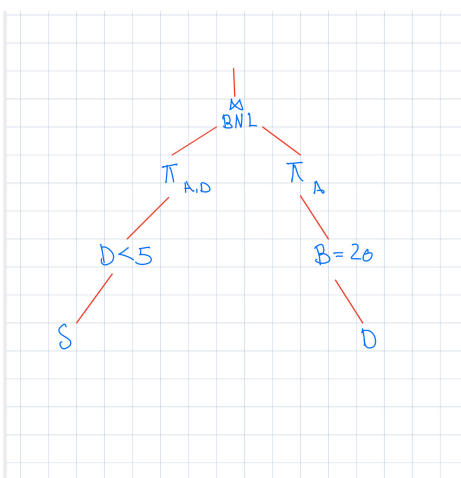
2. What is the size of the result in blocks?

   - the size of row in the result is 10*2 = 20 bytes
   - Number of rows in result = 132
   - Number of rows in block of size 2000 bytes : $\frac{2000}{20}$=100
   - Number if blocks in the result = $\lceil \frac{132}{100} \rceil$ =2

   the size of the result in blocks is 2

3. Draw the query plan tree and specify the most efficient algorithm for calculating the result?

   - Based on various examples in lectures and quizzes, we have seen that it is preferable to use an index scan when there is a choice, especially when selecting rows corresponding to a specific value. However, when searching for rows within a specific range, it is preferable to perform a full range scan on the table.



4. What is the cost of the most efficient calculation?

- we know that
  - Read(E): I/O cost of reading E from disk.
  - B(E): Number of blocks of E.
  - T(E): Number of tuples of E.
  - For block nested loop join, we learned that the cost is: Read(E_R)+Read(E_S)$\times\left\lceil\frac{B(E_R)}{M-2}\right\rceil$
  - we know that M = 70 so M-2 = 68
  - read(S) = B(S) = 1200
  - we know that Number of rows in block of size 2000 bytes : $\frac{2000}{20}$=100
  - Range of D is unknown so Number of rows in result: $\frac{T(S)}{3}$ = $\frac{79200}{3}$ =26,400 rows of S after choosing D<5
  - rows of S after choosing D<5 :Number of rows in block=$\frac{26400}{100}$= 264 Block
  - Number of rows in result assuming uniform distribution of B so Number of rows in result : $\frac{T(R)}{V(R,B)}$
  - read(R) = $\frac{T(R)}{v(R,B)}$= $\frac{400,000}{200}$ =2000 row with value B = 20
  - now Number of rows in block of size 2000 bytes is $\left\lfloor\frac{2000}{10}\right\rfloor$ =200
  - so $\frac{2000}{200}$= 10 Blocks in R == B(R)
  - now we can use the rule that we learned for the cost :

1. Read(E_R)+Read(E_S)$\times\left\lceil\frac{B(E_R)}{M-2}\right\rceil$= 2000+1200*$\left\lceil\frac{10}{68}\right\rceil$ = 3200
2. Read (E_R)+Read(E_S)$\times\left\lceil\frac{B(E_R)}{M-2}\right\rceil$=1200+2000*$\left\lceil\frac{264}{68}\right\rceil$ = 9200

We choose the minimum of them, which is equal to 3200.

QUESTION 3
PART A
Explanation:

1. Subquery for Minimum Duration: The inner subquery (SELECT M2.year, MIN(M2.duration) AS min_duration FROM movies M2 GROUP BY M2.year) computes the minimum duration for each year.

2. Join Condition: We join the movies table M1 with the result of the subquery min_dur on two conditions: M1.year = min_dur.year: Ensures we're matching movies with the correct year. M1.duration = min_dur.min_duration: Matches movies where the duration equals the minimum duration for that year.

3. Steps to Implement and Measure:

4. Save the rewritten query in a file named improved_alternative.sql.

5. Execute the query using explain analyse to obtain the query plan and measure the execution time.

6. Capture a screenshot of the query plan and results similar to the initial question's format.

7. Expected Performance Improvement:

8. Execution Time: This alternative approach may vary in performance compared to the window function method. It leverages a join and aggregation strategy which might be optimized differently by the query planner depending on the database system.

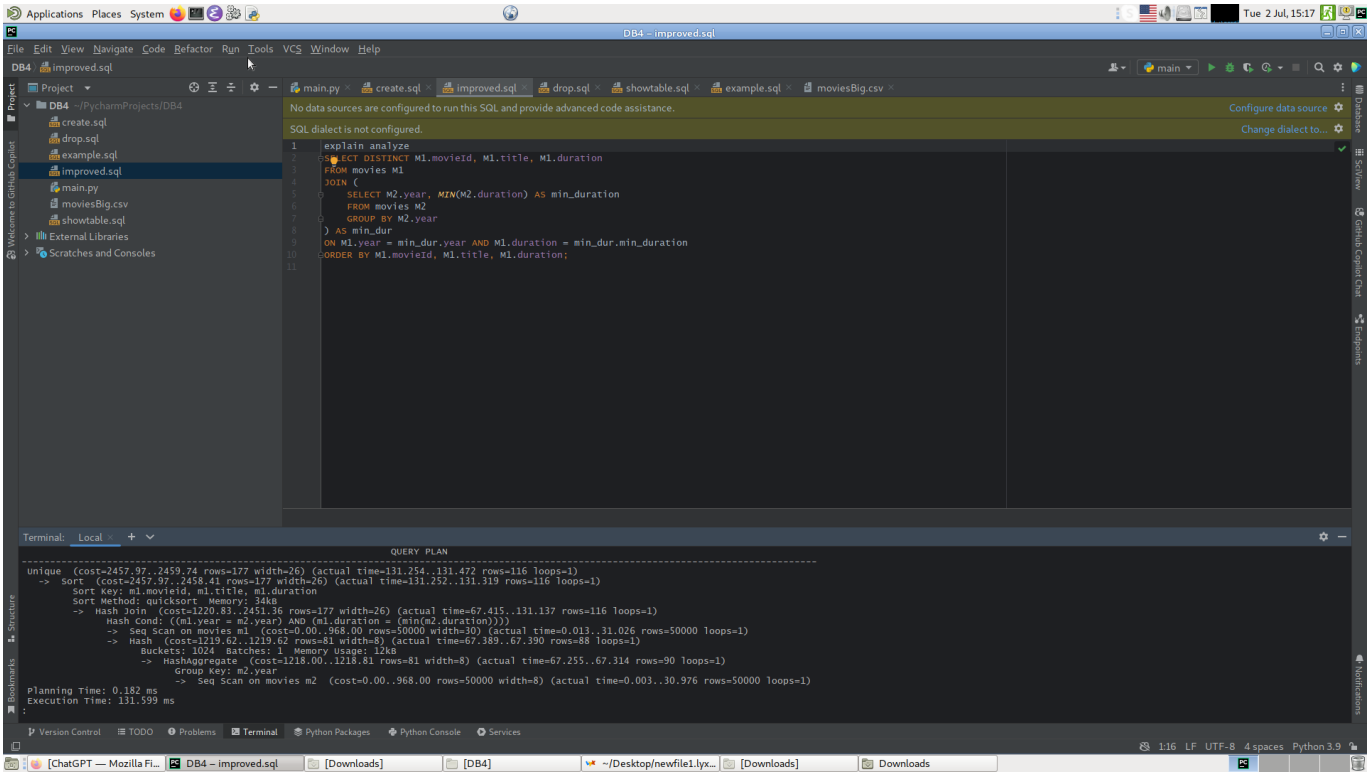
9. Reason for Improvement: By using a join and aggregation instead of correlated subqueries, this approach aims to reduce the number of operations and optimize the query execution plan, potentially improving overall performance.

Based on the QUERY PLAN provided:

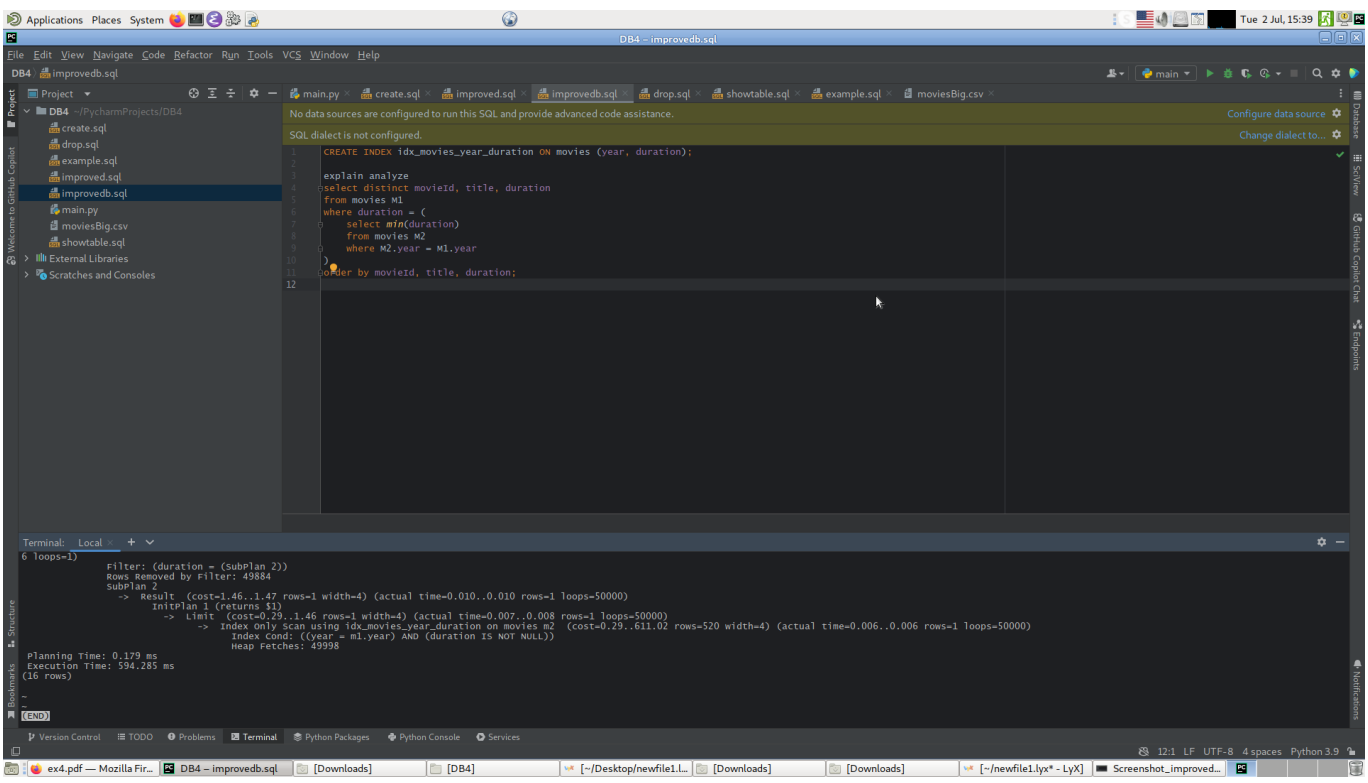
1. Execution Time: The query executed in approximately 131.599 milliseconds.

2. This time includes both planning (0.182 ms) and execution (131.417 ms) phases. Analysis of the Query Plan:

3. Hash Join: This operation joins the movies table m1 with an aggregated result (m2) that computes the minimum duration per year. Hash Aggregate: The inner subquery uses hash aggregation to find the minimum duration per year efficiently. Sorting: A quicksort method is used to sort the results, which contributes to the overall execution time. Memory Usage: The memory used during execution is minimal, indicated by the small memory usage reported


DB4 – improved.sql
explain analyze
SELECT DISTINCT M1.movieId, M1.title, M1.duration
FROM movies M1
JOIN (
SELECT M2.year, MIN(M2.duration) AS min_duration
FROM movies M2
GROUP BY M2.year
) AS min_dur
ON M1.year = min_dur.year AND M1.duration = min_dur.min_duration
ORDER BY M1.movieId, M1.title, M1.duration;
QUERY PLAN
Unique (cost=2457.97..2459.74 rows=177 width=26) (actual time=131.254..131.472 rows=116 loops=1)
-> Sort (cost=2457.97..2458.41 rows=177 width=26) (actual time=131.252..131.319 rows=116 loops=1)
Sort Key: m1.movieid, m1.title, m1.duration
Sort Method: quicksort Memory: 34kB
-> Hash Join (cost=1220.83..2451.36 rows=177 width=26) (actual time=67.415..131.137 rows=116 loops=1)
Hash Cond: ((m1.year = m2.year) AND (m1.duration = (min(m2.duration))))
-> Seq Scan on movies m1 (cost=0.00..968.00 rows=50000 width=30) (actual time=0.013..31.026 rows=50000 loops=1)
-> Hash (cost=1219.62..1219.62 rows=81 width=8) (actual time=67.389..67.390 rows=88 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 12kB
-> HashAggregate (cost=1218.00..1218.81 rows=81 width=8) (actual time=67.255..67.314 rows=90 loops=1)
Group Key: m2.year
-> Seq Scan on movies m2 (cost=0.00..968.00 rows=50000 width=8) (actual time=0.003..30.976 rows=50000 loops=1)
Planning Time: 0.182 ms
Execution Time: 131.599 ms

PART B


DB4 – improvedb.sql
CREATE INDEX idx_movies_year_duration ON movies (year, duration);
explain analyze
select distinct movieId, title, duration
from movies M1
where duration = (
select min(duration)
from movies M2
where M2.year = M1.year
)
order by movieId, title, duration;
Filter: (duration = (SubPlan 2))
Rows Removed by Filter: 49884
SubPlan 2
-> Result (cost=1.46..1.47 rows=1 width=4) (actual time=0.010..0.010 rows=1 loops=50000)
InitPlan 1 (returns $1)
-> Limit (cost=0.29..1.46 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=50000)
-> Index Only Scan using idx_movies_year_duration on movies m2 (cost=0.29..611.02 rows=520 width=4) (actual time=0.006..0.006 rows=1 loops=50000)
Index Cond: ((year = m1.year) AND (duration IS NOT NULL))
Heap Fetches: 49998
Planning Time: 0.179 ms
Execution Time: 594.285 ms
(16 rows)
(END)

- Execution Time: The execution time reported ranges from 593.959 to 594.180 milliseconds. The improvement in execution time is primarily due to the use of the idx_movies_year_duration index in the query, which facilitates easier application of conditions and sorting operations. The larger the table and the more complex the conditions, the greater the performance improvement.

1. Original Query (Without Index): The original query performs a sequential scan (Seq Scan) on the movies table (movies M1). For each row in M1, it executes a correlated subquery (SELECT MIN(duration) FROM movies M2 WHERE M2.year = M1.year) to find the minimum duration for movies in the same year (M1.year). This operation is repeated for each row in M1, resulting in potentially high execution time, especially as the table size increases.

2. Improved Query (With Index idx_movies_year_duration): The improved query benefits from the index idx_movies_year_duration, which is created on the columns year and duration. This index allows the database to efficiently locate rows based on the year and duration criteria without needing to perform a full table scan. When executing the subquery (SELECT MIN(duration) FROM movies M2 WHERE M2.year = M1.year), the index supports quick access to relevant rows in M2, significantly reducing the time required to find the minimum duration per year.