Common Info: Netflix Schema

<table>
<thead>
<tr>
<th>RatingID</th>
<th>Stars</th>
<th>RateDate</th>
<th>UID</th>
<th>MID</th>
</tr>
</thead>
<tbody>
<tr>
<td>7254</td>
<td>4.5</td>
<td>12/15/19</td>
<td>839</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UID</th>
<th>UName</th>
<th>Age</th>
<th>JoinDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Alvarez</td>
<td>39</td>
<td>11/02/14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>492</td>
<td>Parasite</td>
<td>2019</td>
<td>Bong Joon-Ho</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ratings / R
Users / U
Movies / M
• All attributes in the given Netflix database are of fixed length and 8 bytes long, except for Director, Name, and UName, which are 40 bytes long.
• The number of tuples in R, U, and M are 10 billion, 50 million, and 1 million, respectively.
• All tables are stored as heap files with the pages storing the fixed-length records in *unpacked* layout.
• The foreign keys have no dangling references.
• In the given instance, every UID in U and every MID in M arises at least once in R.
Exercise

Q1) Page size is 4096 bytes.

A. [9pts] What is the size of each table in pages? Round to the nearest thousand.

Sizes of tuples: R is 40 B, U is 64 B, and M is 96 B.

NTuples(R) = 10bil, NTuples(U) = 50mil, NTuples(M) = 1mil

Page is 4096 B. Unpacked has bitmap (1b per slot) and count (4 B).

So #tuples/page of R: 4096 \geq 40x + x/8 + 4 \implies \text{max } x \text{ is 101}. So,

#pages of R is \(N_R = \text{ceil}(10 \text{ billion }/ 101) \sim 99,010,000\)

Likewise, we get \(N_U \sim 794,000\) and \(N_M \sim 24,000\)
Exercise

Q1) Page size is 4096 bytes.

B. [3pts] Which 2-table key-foreign key join has the larger output in terms of cardinality? In terms of arity? In terms of size?

2 KFK joins are possible: R JOIN U and R JOIN M.
Both will have the same cardinality (that of R) because neither foreign key has dangling references.
Both will have the same arity because U and M have the same arity and both foreign keys have only one attribute each.
R JOIN M will be larger in size because the attribute set from M is larger than that from U.
C. [5pts] What is the size (in pages) of the output of the 3-table star join in pages? Round to the nearest thousand.

Size of output tuple: 40 + 64 + 96 - 2 * 8 = 184 B
#tuples/page of output: 4096 >= 184x + x/8 + 4 => max x is 22
#tuples of output is same as R: 10 billion
So, #pages of output is ceil(10 billion / 22) ~ 454,545,000
Q1) Page size is 4096 bytes.
D. [8pts] What is the smallest possible size (in pages) of the output of the following query? Round to the nearest thousand.

\[ \pi_{UID}(U) \times \pi_{MID}(M) - \pi_{UID,MID}(R) \]

Size of output tuple: \(2 \times 8 = 16\) B

#tuples/page of output: \(4096 \geq 16x + x/8 + 4\) => max \(x\) is 253

Observe that the crossproduct output cardinality is fixed: \(50m \times 1m = 50,000\) billion. So, to get smallest #tuples possible in the output, we need to check the largest number of tuples from the second project. Clearly, it cannot exceed \(\text{NTuples}(R) = 10\) billion. So, smallest output cardinality is 49,990 billion.

So, #pages of output is \(\text{ceil}(\text{above} / 253) \sim 197,588,933,000\)
Exercise

Q2) Page size is 4096 bytes. You have 1 million buffer frames.

A. [6pts] What is the lowest possible I/O cost (in pages) of sorting \( R \) on Stars using any of the optimizations discussed in the lectures? Round to the nearest thousand.

\( N_R = 99.01 \text{mil.} \) Note output #tuples, #tuples/page, #pages is same.

To get lowest I/O cost, use internal replacement sort but no double buffering or blocked I/O:

\[
2N \left( 1 + \left\lfloor \log_B \left( \left\lceil N/2B \right\rceil \right) \right\rfloor \right)
\]

\[
= 2N \left( 1 + 1 \right) = 396,040,000
\]
Exercise

Q2) Page size is 4096 bytes. You have 1 million buffer frames.

B. [10pts] Suppose page pointers are 8 bytes long and record IDs are 12 bytes long. What is the lowest possible size (in pages) of an extendible hash index built on Users with IndexKey UID using the AltRID alternative? Assume the hash function enables uniform hashing without skews. Round to the nearest thousand. (Hint: Count the number of slots first).

IndexKey is primary key of U. So, #slots = NTuples(U) = 50mil

Slot size = 8 b (UID) + 12 b (record ID) = 20 b; 4B for LD

#slots/bucket = floor((4096 - 4)/20) = 204

#buckets = ceil(50mil/204) = 245,099

Dir has 1 page pointer per bucket; #pointers/page = 4096/8 = 512; so #dir pages = ceil(245099/512) = 479

So, total #pages = 245,099 + 479 ~ 246,000.
Exercise

Q3) Page size is 4096 bytes. You have 500,000 buffer frames for each query (independent of the others). What is the lowest possible I/O cost (in pages, rounded to the nearest thousand) of each of the following queries regardless of the data distributions using only the operator implementations discussed in the lectures? Include the cost of writing the output in C and D.

A. [6pts] \( \pi_{\text{JoinDate}}(U) \)

\( N_U = 794,000 \). Without extra histogram info on JoinDate, assume dedup count worst case of NTuples(U). So, worst-case non-dedup intermediate \( N_T \sim (8/64) N_U \sim 99,250 \). Since T fits entirely in DRAM, overall I/O cost (excluding output write cost) is just \( N_U = 794,000 \).
Exercise

Q3) Page size is 4096 bytes. You have 500,000 buffer frames for each query (independent of the others). What is the lowest possible I/O cost (in pages, rounded to the nearest thousand) of each of the following queries regardless of the data distributions using only the operator implementations discussed in the lectures? Include the cost of writing the output in C and D.

B. [8pts] \( \gamma_{Director, COUNT(\star)}(M \bowtie R) \)

\( N_R = 99,01\text{mil}; N_M = 24,000; B = 0.5\text{mil} \sim 2 \text{ GB} \)

Clearly a hash join is feasible with just one pass over each base table. Pipeline it to a hash-based group by. #entries in hash table in worst-case is NTuples(M) = 1mil. Entry size is 40 b + 8b = 48b. So, max hash table size is just 1.4 * 1mil * 48 b \sim 68 \text{ MB}, well under B. So, whole PQP is fully pipelined and no partitioning is needed anywhere. Overall I/O cost is just one read of each base table: \( N_R + N_M = 99,034,000 \).
Exercise

Q3) Page size is 4096 bytes. You have 500,000 buffer frames for each query (independent of the others). What is the lowest possible I/O cost (in pages, rounded to the nearest thousand) of each of the following queries regardless of the data distributions using only the operator implementations discussed in the lectures? Include the cost of writing the output in C and D.

C. [10pts] \( R \Join U \Join M \)

\( N_R = 99,01\text{mil}; N_U = 794,000; N_M = 24,000; B = 0.5\text{mil} \sim 2 \text{ GB} \)

Clearly \( R \Join U \) cannot execute with just one read of each base table because \( U \) is larger than available RAM. HJ or SMJ will have same cost based on given \( B \). But \( R \Join M \) can run with just one read of \( M \) because \( M \) is much smaller than available RAM. So, do a PQP with right-deep tree with \( M \) as leftmost joined with the output of \( U \) joined with \( R \) as rightmost. Fully pipelined plan with join I/O cost = \( N_M + 3(N_U + N_R) = 299,436,000 \).

Output size for write is 454,545,000. So, total I/O cost is \( 753,891,000 \).
Exercise

Q3) Page size is 4096 bytes. You have 500,000 buffer frames for each query (independent of the others). What is the lowest possible I/O cost (in pages, rounded to the nearest thousand) of each of the following queries regardless of the data distributions using only the operator implementations discussed in the lectures? Include the cost of writing the output in C and D.

D. [10pts] \( \pi_{UID}(U) \times \pi_{MID}(M) \)

Output tuple size: \( 2 * 8 = 16 \) B; So, #tuples/page of output: 4096 >= 16x + x/8 + 4 => max x is 253; Output cardinality: 50m x 1m = 50,000 billion; So, #pages is ceil(above / 253) ~ \( 197,628,458,500 \)

Use BNLJ for crossproduct. \( N_U = 794,000; N_M = 24,000; B = 0.5 \text{mil} \sim 2 \text{ GB} \)

So, smaller table M can fit entirely as a single block in DRAM. Project on UID and MID implicitly do not need dedup, since they are primary keys. So, whole crossproduct needs only 1 read of M entirely into DRAM and 1 streaming read of U. It adds a small \( N_U + N_M \) to the above.

Total I/O cost: \( 197,629,276,500 \).
Q4) Are these pairs of queries equivalent? If you say yes, explain why. If you say no, provide a counterexample.

A. [5pts] 

\[
\pi_{\text{UID}}(U) \times \pi_{\text{MID}}(M) \\
\pi_{\text{UID, MID}}(U \times M)
\]

Yes. The schemas are same. The output cardinality will always be the same, viz., NTuples(U) * NTuples(M), since these are the primary keys. The output will have all pairs of UIDs and MIDs from U and M on the left and on the right.
Exercise

Q4) Are these pairs of queries equivalent? If you say yes, explain why. If you say no, provide a counterexample.

B. [6pts] \[ \gamma_{Year, \text{MAX}(Stars)}(R \bowtie M) \]
\[ \gamma_{Year, \text{MAX}(Stars)}(M \bowtie \gamma_{\text{MID}, \text{MAX}(Stars)}(R)) \]

Yes. The schemas are same. The output cardinality will always be the same, viz., \#values of Year in M.

Now the partial pre-aggregation in R in the second query reduces the cardinality to \#unique values of MID in R. But MID is the join attribute. So, the join cardinality is unaffected. Multiple MIDs might have the same Year in M. But due to the property of MAX, the MAX taken over MIDs in the second query is topped up with a MAX taken over Year. MAX of multiple MAX values corresponding to the same Year (for each MID) will still give the same global MAX for each Year as the query above.
Exercise

Q4) Are these pairs of queries equivalent? If you say yes, explain why. If you say no, provide a counterexample.

C. [6pts] \[ \gamma \text{COUNT}(\star)(R \bowtie U \bowtie M) \]
\[ \gamma \text{COUNT}(\star)(\pi \text{UID, MID}(R) \bowtie U \bowtie M) \]

No. Schemas are same. Output cardinality after group by (1) are same.
But in the second query, the deduplication of R on (UID,MID) may alter the star join’s intermediate output cardinality before the group by because duplicates of (UID,MID) across tuples are possible in R. Thus, the output instance values may differ.
Simplest counterexample: R has [(RID = 1, …, UID = 1, MID = 1), (RID = 2, …, UID = 1, MID =1)]. U has [(UID = 1, …)]. M has [(MID = 1, …)]. The first query gives a count of 2; the second gives a count of 1.