Exercise 4

Time tip: Roughly 45sec to 1min per 1pt
### Ratings / R

<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>

### Users / U

<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>

### Movies / M

<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>

---

**Common Info: Netflix Schema**
Exercise

Q1) [4 x 5pts] You are given two instances of R (R1 and R2) and the following statistics of the number of pages in each relation and the allotted buffer memory size in pages B. Page size is 8 KB. Suppose all attributes are 8 bytes long. Assume UID and MID are uniformly distributed in R. Ignore output write costs. What is the I/O cost (in number of pages) of the following operations using any of the implementations discussed in the lecture?

\[(N_{R1}, N_{R2}, B) = (40000, 25000, 5000)\]

A. Intersection of R1 and R2

B. Union of R1 and R2

C. Set difference R1 - R2

D. Set difference R2 - R1

A. Use regular hash join. \(B^2 > FN_{R2}\) satisfied. So, I/O cost is \(3(N_{R1} + N_{R2}) = 195,000\).

B. Use the hash-based impl. The partitioning stage like in HJ ensures hash table on split \(i\) will be under \(B\) even after unioning \(i\)’th splits of both tables. So, 195,000 again.

C. Similar to B; 195,000.

D. Also similar to B; 195,000.
Q2) [3 x 4pts] You are given the following statistics of the number of pages in U and the allotted buffer memory size in pages B. Suppose all attributes are 8 bytes long, except U.Name, which is 40 bytes. Page size is 8 KB. You are also given a clustered AltRID B+ tree index on U with IndexKey (JoinDate, Age). RID length is also 8 bytes. What is the rough I/O cost (in number of pages) of the following operation with the specified implementation?

A. Hashing-based aggregate
B. Sorting-based aggregate
C. Index-based aggregate

(N_u, B) = (10000, 500)

\( \gamma \text{COUNT}(\text{DISTINCTAge})(U) \)

C. Attributes need for group by are subset of IndexKey but not prefix. So, read leaf level as input for hash/sort-based group by. A data entry has JoinDate, Age, and RID; so, leaf level size is roughly \( (24/64) \) \( N_u = 3750 \). So, 3750 + 2*1250 = 6,250.

A. Read U and build hash table on Age column to deduplicate (kind of like a project) and then count. Non-deduplicated Age column size \( N_T = (8/64)N_U = 1250 \). So, hash table may not fit in RAM, requiring partitioning. So, \( N_U + 2N_T = 12,500 \).

B. Likewise, \( N_U + 2N_T = 12,500 \).
Exercise

Q3) [3 x 4pts] You are given the following statistics of the number of pages in U and the allotted buffer memory size in pages B. Suppose all attributes are 8 bytes long, except U.Name, which is 40 bytes. Page size is 8 KB. You are also given a clustered AltRID B+ tree index on U with IndexKey (JoinDate, Age). RID length is also 8 bytes. What is the rough I/O cost (in number of pages) of the following operation with the specified implementation?

A. Hashing-based aggregate 
B. Sorting-based aggregate 
C. Index-based aggregate 

\((N_U, B) = (10000, 500)\)

\(\gamma_{JoinDate, AVG(Age)}(U)\)

C. Attributes need for group by are subset of IndexKey, while grouping list is also prefix of IndexKey. So, just one seq. read of leaf level suffices to compute incr. stats for avg. Leaf level size is roughly \((24/64) N_U = 3750\). So, total I/O cost is roughly just 3,750.

A. Build hash table on JoinDate column with 2 running stats for avg of Age. Max size of hash table is \(F*(3*8/64)*N_U\), which is clearly > B. So need to write hash-partitions of T(JoinDate,Age). \(N_T = (2*8/64)*N_U\). So, total is \(N_U + 2N_T = 15,000\).

B. Likewise, \(N_U + 2N_T = 15,000\).
Exercise

Q4) [3 x 4pts] You are given the following statistics of the number of pages in U and the allotted buffer memory size in pages B. Suppose all attributes are 8 bytes long, except U.Name, which is 40 bytes. Page size is 8 KB. You are also given a clustered AltRID B+ tree index on U with IndexKey (JoinDate, Age). RID length is also 8 bytes. What is the rough I/O cost (in number of pages) of the following operation with the specified implementation?

A. Hashing-based aggregate
B. Sorting-based aggregate
C. Index-based aggregate

\( \gamma \text{COUNT(DISTINCT Age)}(U) \)

\((N_U, B) = (10000, 7000)\)

A. Just like Q2.A, non-deduplicated Age column size \(N_T = (8/64)N_U = 1250\). But now hash table on it fits in RAM; so no need for partitioning and we need just a read of U. So, 10,000.

B. Likewise, we can sort the non-dedup Age column in RAM. So, 10,000.

C. Like Q2.C, read leaf level as input for hash/sort-based group by. Leaf level size is roughly \((24/64) N_U = 3750\), which fits in RAM entirely. So, we need just one read of leaf level. So, roughly just 3,750.
Q5) [3pts] Which of the following relational equivalencies hold?

A  \( R \cup R = R \)

B  \( R \cap R = R \)

C  \( R \bowtie R = R \)

D  All of A, B, C

E  None of the other options
Exercise

Q6) [4pts] Which of the following relational equivalencies hold?

A  $\pi_A(R \times S) = \pi_{A \cap R.\star}(R) \times \pi_{A \cap S.\star}(S)$

B  $\pi_A(R \bowtie S) = \pi_{A \cap R.\star}(R) \bowtie \pi_{A \cap S.\star}(S)$

C  Both A and B

D  None of the other options

B does not hold because the schemas may not even be the same! Note that joining attribute(s) may not be a subset of A, which means they may be lost if project is pushed down to the base tables this way, resulting in the RHS join degenerating into a crossproduct.
Q7) [5pts] Which of the following relational equivalencies hold?

A $\gamma_{A, COUNT(\star)}(R) = \gamma_{A, COUNT(\star)}(\pi_A(R))$

B $\gamma_{MAX(B)}(R) = \gamma_{MAX(B)}(\pi_B(R))$

C $\gamma_{SUM(B)}(R) = \gamma_{SUM(B)}(\pi_B(R))$

D All of A, B, C

E None of the other options

A and C do not hold because the project on $R$ will remove any duplicates, thus potentially altering the results of the COUNT / SUM outside. MAX (and MIN) is unaffected by such deduplication.
Exercise

Q8) [5pts] Which of these queries has/have at least 6 possible PQPs based only on the physical operators we saw in class?

A  \( \sigma_{Stars > 4}(R \Join M) \)

B  \( \pi_{Name}(\sigma_{Stars > 4}(R \Join M)) \)

C  \( \pi_{Name}(R \Join M) \)

D  All of A, B, C

E  None of the other options

A has \( >= 2 \) (select) x \( 3 \) (join)
B has \( >= 2 \) (project) x \( 2 \) x \( 3 \)
C has \( >= 2 \) x \( 3 \)

D  All of A, B, C

E  None of the other options
Q9) You are given the following statistics of the number of pages of each relation in the Netflix database shown. Suppose all attributes are 8 bytes long, except U.Name, M.Name, and M.Director, each of is 40 bytes. Assume UID and MID are uniformly distributed in R. Ignore output write costs. Page size is 8 KB. What is the lowest estimate possible of the largest size of the output table of the following query (in # pages) with only the given information? \((N_R, N_U, N_M, B) = (80000, 20000, 5000)\)

A. [4pts] \(\pi_{UID, MID}(R)\)

A. To compute the size of a table, we need its cardinality and tuple size. Clearly, the cardinality of this output cannot exceed NTuples(R). The size of a tuple of R is 5 * 8 bytes = 40 bytes. The size of an output tuple here is 2 * 8 bytes = 16 bytes. So the ratio is 16/40. If we rescale \(N_R\) with this ratio, we will get the rough largest number of pages of output. So, the answer is \((16/40) \times N_R = 32,000\).
Q9) You are given the following statistics of the number of pages of each relation in the Netflix database shown. Suppose all attributes are 8 bytes long, except U.Name, M.Name, and M.Director, each of is 40 bytes. Assume UID and MID are uniformly distributed in R. Ignore output write costs. Page size is 8 KB. What is the lowest estimate possible of the largest size of the output table of the following query (in # pages) with only the given information? \((N_R, N_U, N_M, B) = (80000, 20000, 5000)\)

B. [4pts] \(\gamma_{Director, \text{AVG}(Stars)}(M \bowtie R)\)

B. The cardinality of this output cannot exceed NTuples(M). The size of a tuple of the output is 40 + 8 = 48 bytes (AVG column can be a double precision floating point). The tuple size of M is (8 + 40) * 2 = 96 bytes. So, the answer is \((48/96) \times N_M = 2,500\).
Exercise

Q9) You are given the following statistics of the number of pages of each relation in the Netflix database shown. Suppose all attributes are 8 bytes long, except U.Name, M.Name, and M.Director, each of is 40 bytes. Assume UID and MID are uniformly distributed in R. Ignore output write costs. Page size is 8 KB. What is the lowest estimate possible of the largest size of the output table of the following query (in # pages) with only the given information? (N_R, N_U, N_M, B) = (80000, 20000, 5000)

C. [6pts] \( R \bowtie U \bowtie M \)

B. The cardinality of this output cannot exceed NTuples(R), since both joins are key-foreign key joins in a star schema where R is the fact table. The size of a tuple of R is 5 * 8 = 40 bytes. The output is a concatenation of the schemas of all 3 tables, except since these are natural joins, UID and MID will occur only once in the output. So, the output tuple size is 40 (R) + 64 (U) + 96 (M) - 2 * 8 (redundant UID, MID) = 184 bytes. So, the answer is \((184/40) * N_R = 368,000\).
Exercise

Q10) You are given the following statistics of the number of pages of each relation in the Netflix database shown and the allotted buffer memory size in pages \( B \). Suppose all attributes are 8 bytes long, except U.Name, M.Name, and M.Director, each of is 40 bytes. Assume UID and MID are uniformly distributed in \( R \). Ignore output write costs. Page size is 8 KB. No indexes exist in the database.

\[
(N_R, N_U, N_M, B) = (50000, 10000, 2000, 5000)
\]

A. [4pts] Which key-foreign key join in this database can NOT be executed using just one read of each base table?

A. There are only 2 KFK joins: \( R \ JOIN \ U \) and \( R \ JOIN \ M \). A hash join for \( R \ JOIN \ M \) can be executed with just one read of each base table, since \( N_M < B \). The other join cannot be executed this way, since the smaller table is \( > B \).
Exercise

Q10) You are given the following statistics of the number of pages of each relation in the Netflix database shown and the allotted buffer memory size in pages $B$. Suppose all attributes are 8 bytes long, except U.Name, M.Name, and M.Director, each of is 40 bytes. Assume UID and MID are uniformly distributed in R. Ignore output write costs. Page size is 8 KB. No indexes exist in the database.

$$(N_R, N_U, N_M, B) = (50000, 10000, 2000, 5000)$$

$\gamma COUNT(*)(\sigma UID=123(R \bowtie U))$

B. [6pts] Propose a fully pipelined PQP for this query that can be executed with just one scan of each base table.

Group by COUNT(*)

B. Push down select to both base tables; even to just U suffices

Hash join

Filescan w/ check UID = 123

Filescan w/ check UID = 123

U

R
Exercise

Q10) (Continued) \((N_R, N_U, N_M, B) = (50000, 10000, 2000, 1000)\)

C. **[8pts]** What is the lowest possible I/O cost of this query using only the operator implementations discussed in the lectures? Consider possible algebraic rewrites too. Note the lower B. Explain your approach in detail.

\[ \gamma_{Year, \text{COUNT}}(\ast)(R \bowtie M) \]

C. (Also see the associated discussion video)

Clearly, the lower bound on I/O cost is \(N_R + N_M\). So, let us first try to produce a PQP that is fully pipelined and achieves this cost.

We have smaller table \(N_M > B\). So, hash join will need partitioning. The other join impl. will also have similar cost and not reach the lower bound.

But observe the second operator in the LQP is a group by that only needs \(Year\). So, the join really does NOT need to send out all attributes of \(M\). This motivates a logical rewrite of the LQP wherein we introduce a non-deduplicating project on \(M\) before the join to retain only attributes needed for the join and group by, viz., \((MID, Year)\). This is still correct because MID is a key in \(M\) and the project will not change the join output cardinality, which means the count results will be preserved. (Continued next slide …)
C. So, the overall new LQP is as follows, wherein \( \pi^* \) denotes a non-dedup. project

\[
\gamma_{Year,COUNT(*)}(R \bowtie M)
\]

To decide the operator implementations to pick for the PQP, we need to ensure the B given is enough to ensure a fully pipelined execution with just one read of M and R.

The non-dedup project intermediate output is of size \((16/96)*2000 \sim 334\). This easily fits in RAM as a hash table of size \(1.4*334 \sim 468\). So, choose hash-based join with MID as the hashing key and the buckets having (MID,Year). Since the hash table fits in RAM, we need only one streaming read of R for the join. Pipeline the join output to a hash-based group by at the end with a hash table with hashing key Year and the incremental count statistic in the bucket too. The max size of this second hash tables is also just \(1.4*(16/96)*2000 \sim 468\). So, both hash tables fit together in RAM. The whole query needs no partitioning of inputs nor does any intermediate need to be written to disk for any operator!

Thus, overall we got a PQP that hits the lower bound I/O cost for this query, viz., \( N_M + N_R = 52,000 \).
Exercise

Q10) (Continued) \[(N_R, N_U, N_M, B) = (50000, 10000, 2000, 1000)\]

D. **[10pts]** What is the lowest I/O cost of this query using only hash joins? Note the lower B. Explain your approach in detail.

\[ R \bowtie U \bowtie M \]

D. (Also see the associated discussion video)

Clearly we cannot hit the lower bound I/O cost of just 1 read of all tables because the smallest table (M) itself is > B. We also know right deep is amenable for pipelining with HJ. What we need to figure out is the cost of producing the intermediates for different right deep tree plans. We must pair (R,U) or (R,M), since only they share attributes for a natural; note (U,M) degenerates to a crossproduct. We must also put the smaller table on the left of a HJ. So, we need only consider the following right deep tree plans:

![Diagram of right deep tree plans]

In either plan, we need to write the temporary output of the first join to disk and re-partition it for the second join because the hashing attribute for the second join is different: MID first, then UID for Tree1 and vice versa for Tree2. So, now we need to figure out which leads to a smaller intermediate table.
Q10) (Continued) \((N_R, N_U, N_M, B) = (50000, 10000, 2000, 1000)\)

D. **[10pts]** What is the lowest I/O cost of this query using only hash joins? Note the lower B. Explain your approach in detail.

\[
R \bowtie U \bowtie M
\]

The intermediate in Tree1 is a full natural join of M and R. What is its size? We know it is a KFK join. So, its cardinality is upper bound by \(NTuples(R)\). This means in the worst case, we add the extra attributes of M to R and rescale it. In this temporarily table \(T\), a tuple has size: 40 from \(R\) + 96 from \(M\) - 8 duplicate \(MID = 128\) bytes. So, we have roughly \(N_T = (128/40) * N_R = 160,000\).

Similarly, the intermediate table in Tree2 is roughly \(N_T = (96/40) * N_R = 120,000\). Thus, Tree2 will result in a lower I/O cost overall. How to get its total I/O cost?

For the first HJ, we need a partitioning on \(UID\); this cost is the usual \(3(N_U + N_R)\). The intermediate output \(T\) is pipelined to a new hash partitioning for the second HJ wherein partitioning is on \(MID\). So, we have 1 write and 1 read of \(T\), and 1 partitioning and 1 read of \(M\). So, total cost is:

\[
3N_U + 3N_R + N_T + N_T + 3N_M = 426,000.
\]