# Common Info: Netflix Schema

## 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>Name</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>

---

**Ratings / R**

- **RatingID**: Unique identifier for each rating.
- **Stars**: Rating value from 1 to 5 stars.
- **RateDate**: Date of the rating.
- **UID**: Unique identifier for the user who rated the movie.
- **MID**: Unique identifier for the movie.

**Users / U**

- **UID**: Unique identifier for each user.
- **Name**: Name of the user.
- **Age**: Age of the user.
- **JoinDate**: Date the user joined the platform.

**Movies / M**

- **MID**: Unique identifier for each movie.
- **Name**: Name of the movie.
- **Year**: Year of release.
- **Director**: Name of the director.
Exercise

Q1) [2 x 2pts] Suppose you are given that NTtuples(R) = 1 billion. What is the selectivity of a selection predicate on R if it results in the following number of tuples in the output?

A. 50
B. 50 million

A. \frac{50}{1 \text{ billion}} = 0.000005 \% 
B. \frac{50 \text{ million}}{1 \text{ billion}} = 5\%
Exercise

Q2) [4 x 6pts] Name 3 indexes each (at least 1 each of B+ tree and hash index, if possible) that match the following select operations.

A. SELECT * FROM R WHERE UID = 32 AND Stars >= 4

B. SELECT * FROM M WHERE NOT (Year < 2010 OR Year >= 2020)

C. SELECT * FROM U WHERE NAME LIKE ‘A%’ AND Age >= 30 OR NAME LIKE ‘B%’ AND Age >= 30

D. SELECT * FROM R WHERE NOT (UID <> 8 OR MID = 9)

A. Clustered B+ tree on UID / Stars; hash on UID

B. Clustered B+ tree on Year / (Year, MID) / (Year, Name); no matching hash index

C. Clustered B+ tree on Age / (Age, UID) / (Age, Name); no matching hash index

D. Clustered B+ tree on UID / (UID, Stars); hash on UID
Q3) [6 x 5pts] You are given the following statistics of the number of pages in each relation 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. What is the I/O cost (in number of pages) of the following operators?

\((N_R, N_U, N_M, B) = (20000, 4000, 1000, 500)\)

A. Hash-based project of R on to RateDate
B. Sort-based project of R on to RateDate
C. Hash-based project of U on to Name
D. Sort-based project of U on to Name
E. Hash-based project of M on to Director
F. Sort-based project of M on to Director

A. 28,000
B. 28,000
C. 9,000
D. 9,000
E. ~1,833
F. ~1,833
Exercise

Q4) [6 x 5pts] You are given the following statistics of the number of pages in each relation 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. What is the I/O cost (in number of pages) of the following operators?

\[(N_R, N_U, N_M, B) = (40000, 5000, 500, 800)\]

A. BNLJ for natural join of U and R
B. SMJ for natural join of U and R
C. HJ for natural join of U and R
D. BNLJ for natural join of M and R
E. SMJ for natural join of M and R
F. HJ for natural join of M and R

A. 365,000
B. 135,000
C. 135,000
D. 40,500
E. 121,500
F. 40,500
Q5) [2 x 6pts] You are given the following statistics of the number of pages in each relation. 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. Suppose the allowed buffer memory size in pages is $B$. Identify a value of $B$ for which BNLJ will have a lower I/O cost than HJ or SMJ for the following join.

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

A. Natural join of U and R
B. Natural join of M and R

A. We need $B$ s.t. HJ needs partitioning but BNLJ needs at most 3 passes over R, i.e., need $\lceil FN_U / (B - 2) \rceil = 2$ or $3$ but not $1$ or $4$; so, $B \sim 4669$ to $\sim 14001$

B. Similarly, $B$ can be $\sim 936$ to $\sim 2801$
Exercise

Q6) [10pts] You have \(4B + 1\) buffer frames for the following hash join of \(R\) and \(U\), which have sizes \(16*B*r\) and \(4*B*u\) pages, respectively. You also have \(16Br \ll 4Bu\). The buffer pool is initially empty. You are also given that \(3Fr = 4B - 1\), where \(F\) is the hash table fudge factor. \(B\) is in the millions.

The distribution of UID in \(R\) is such that after the first hash partitioning phase, we get exactly \(4B\) partitions each of \(R\) and \(U\). Each partition of \(U\) is \(u\) pages long, but the partitions of \(R\) have differing lengths.

Suppose \(R\) gets partitioned as follows: \(B\) partitions of length \(r\) pages each and \(3B\) partitions of length \(5r\) pages each. What is the I/O cost of the above join using the regular hash join discussed in class? Exclude the cost of writing the output. Assume that perfect uniform splitting occurs during the recursive repartitioning and that we do not need to recurse more than once.

The smaller table (\(R\)) does not fit in buffer memory as a hash table. So, we need partitioning. This makes the I/O cost of this hash join at least \(3 \times (16Br + 4Bu) = 48Br + 12Bu\).

But one round of partitioning does not suffice, since buffer memory is not enough to fit the hash table during the stitching phase for the \(3B\) partitions of \(R\) that are of length \(5r\) pages each. So, we need a round of recursive re-partitioning for those. This in turn means recursive re-partitioning for the corresponding partitions of \(U\) as well. We are given that we need not recurse further.

So, the I/O goes up further by \(1\) read \(+\) \(1\) write of these \(3B\) partitions of \(R\) and \(U\), which adds a total of \(2 \times (3B \times 5r + 3B \times u) = 30Br + 6Bu\).

Thus, the overall I/O cost (excluding output write) works out to: \(78Br + 18Bu\)
**Exercise**

**Q7) [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.
Q8) [3 x 5pts] 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

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

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

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, \(3750 + 2 \times 1250 = 6250\).

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

Q9) [3 x 5pts] 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

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

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

C. Like Q8.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.

A. Just like Q8.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.
Q10) [3 x 5pts] 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)
\
\left< JoinDate, AVG(Age) \right>_U

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.

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.