Q 1. [20pts] For the following questions, clearly circle True or False.

1. Rename is the only operator in relational algebra whose semantics depends only on the database schema and not the database instance.
   True

2. Key-foreign key joins are always inner joins.
   True

3. It is always possible to store variable length records in the packed page layout.
   False

4. All clustered indexes are primary indexes.
   False

5. Internal replacement sort sometimes reduces the number of passes needed for the merge phase of external merge sort.
   True

6. Given a physical query plan, there is only one logical query plan that can be translated to it.
   False

7. Most RDBMSs use the independence heuristic for estimating the selectivity of a conjunction of predicates.
   True

8. It is sometimes possible to obtain superlinear speedup in a parallel DBMS.
   True
9. Fault tolerance is one of the key reasons for the success of “Big Data” systems such as MapReduce/Hadoop and Spark compared to parallel DBMSs.

True

10. Variance is not an algebraic aggregate.

False

Q 2. [32pts] For the following questions, clearly circle the right answer (only one option is correct).

1. Which of the following relational operators usually can not be sped up (in terms of I/O costs) with a B+ tree index at all?

   (a) σ  (b) π  (c) γ  (d) ×  (e) ◁

   ANSWER: (d)

2. Given a relation \( R(A, B, C) \), which of the following indexes do not match the following query: \( \sigma_{A=a, B=b}(R) \)?

   (a) Hash index on \( R(B) \)  (b) Hash index on \( R(A, B) \)  (c) Hash index on \( R(A, C) \)

   ANSWER: (c)

3. Which of the following paradigms of parallelism is followed by most parallel DBMSs and “Big Data” systems?

   (a) Shared-memory  (b) Shared-nothing  (c) Shared-everything  (d) Shared-disk

   ANSWER: (b)

4. A shuffle is never needed for which of the following relational operators in a parallel DBMS?

   (a) σ  (b) π  (c) γ  (d) ×  (e) ◁

   ANSWER: (a)
5. Which of the following physical join operators is a blocking operator?

(a) Block nested loop  (b) Sort merge  (c) Hash  (d) Index nested loop

**ANSWER: (b)**

6. Which is the most common data partitioning scheme in parallel DBMSs?

(a) Random  (b) Range-based  (c) Hashing-based  (d) Round Robin

**ANSWER: (c)**

7. How many different hash functions does the two-phases “improved” hash join implementation in a parallel DBMS use?

(a) 1  (b) 2  (c) 3  (d) 4  (e) 5

**ANSWER: (c).** One for partitioning, one for repartitioning, and one for hash table. But there seems to have been some ambiguity in the question on whether the partitioning should be included. So, (b) is also an acceptable answer.

8. Column stores were introduced to speed up primarily which of these relational operations?

(a) Aggregate  (b) Intersection  (c) Set difference  (d) Join

**ANSWER: (a)**

9. In practice, what is the minimum percentage occupancy of a page typically maintained by a B+ tree index?

(a) 0  (b) 25%  (c) 50%  (d) 75%  (e) 99%

**ANSWER: (c)**

10. Which of the following invariants/conditions *always* hold for an extendible hash index? GD and LD stand for global and local depths respectively.

(a) All LD < GD  (b) All LD ≤ GD  (c) All LD > GD  (d) All LD ≥ GD
ANSWER: (b)

11. Which of the following improvements of external merge sort does not affect the fan-in of the merge phase?

(a) Internal replacement sort  (b) Double buffering  (c) Blocked I/O
ANSWER: (a)

12. Which typical system resource in a parallel DBMS is not underutilized by a multi-way external merge sort compared to the simple 2-way variant?

(a) Disk  (b) RAM  (c) CPU  (d) Processor caches  (e) Network
ANSWER: (b)

13. Which of the following SQL capabilities have no counterpart in (extended) relational algebra?

(a) SELECT DISTINCT  (b) NATURAL JOIN  (c) GROUP BY  (d) ORDER BY
ANSWER: (d)

14. Which buffer replacement policy is a variant of LRU with lower overhead?

(a) MRU  (b) Clock  (c) Random  (d) FIFO
ANSWER: (b)

15. Which of the following physical join operators benefits when both tables being joined are already stored in the order of the joining attributes’ domain?

(a) Block nested loop  (b) Sort merge  (c) Hash  (d) Index nested loop
ANSWER: (b)

16. Which typical system resource is not underutilized by the block nested loop join compared to the simple page-nested loop join?
Q 3. [10pts] Operator Implementation. We are given two tables $R(X, Y)$ and $S(X, Y)$ with the same schema, i.e., they are union-compatible. Assume attributes $X$ and $Y$ are of the same size. The table sizes of $R$ and $S$ happen to be $1.2B$ and $3B$ pages respectively, where $B$ is the number of buffer pages available (typically, in the millions). Assume there are no skews in any attribute distributions. Hash table fudge factor is 1.4.

Given the above, what is the smallest I/O cost for each of the following operators in terms of $B$? Exclude the cost of writing the output. Clearly mention the physical operator implementation you chose and briefly explain your calculation of the answer without skipping key steps; otherwise, no points will be given!

1. (2pts) $\pi_X(R)$

**Answer:** $1.2B$. Choose sort-based project. Since $X$ and $Y$ are of same length, temporary non-deduplicated file is of size roughly $0.6B$, which fits in memory for a full internal sort. So, we only need one read of $R$. A hash-based project also has the same I/O cost, since the hash table on the temporary file with only $X$ is of size roughly $F|R|/2 \approx 0.84B$, which also fits in memory.

2. (4pts) $R \cap S$

**Answer:** $7.2B$. A regular join implementation suffices to compute intersection, wherein we do an equi-join on both $X$ and $Y$. We can use BNLJ. $R$ is the smaller outer table. The general I/O cost is $|R| + \lceil (F \cdot |R|/(B - 2)) \rceil \cdot |S| = 1.2B + \lceil (1.4 \cdot 1.2B/(B - 2)) \rceil \cdot 3B$. Since $B$ is typically in the millions, this works out to $|R| + 2|S| = 7.2B$. Note that a hash join requires partitioning $R$ and $S$ into 2 splits each, since hash table on all of $R$ does not fit in memory. Thus, overall I/O cost of hash join is $3(|R| + |S|) = 12.6B$, which is larger than the cost of BNLJ above. Sort-merge join cost will be even larger.

3. (4pts) $R \times S$

**Answer:** $7.2B$. BNLJ suffices for cross product too, except there is no join predicate for it. Same cost analysis as in the previous answer applies, and BNLJ has the smallest I/O cost again.
Q 4. [20pts] Query Processing. Recall the Netflix schema discussed in class:

\[ R(RID, Stars, RateDate, UID, MID) \]
\[ U(UID, UName, Age, JoinDate) \]
\[ M(MID, MName, Year, Director) \]

R.MID is a foreign key referring to M.MID; R.UID is a foreign key referring to U.UID

We are given the following pieces of information about the database and the system: \( N_R = 10^7 \), \( N_U = 10^6 \), and \( N_M = 10^5 \) (respective number of pages of each table), page size of 8 KB, and allotted buffer memory of 10 GB. Assume that the length of each attribute is 8 B, except for UName, MName, and Director, each of which is of length 40 B. The fudge factor for hash tables is 1.4.

Answer the following questions. Clearly and briefly explain your derivation/calculation of the answer by mentioning the key steps, including the logical query plan, the chosen physical query plan, whether each operator is pipelined or materialized, and explaining how the I/O costs add up to the given number. Otherwise, no points will be awarded!

1. (6pts) Name 3 indexes, with at least one of them being a hash index, that match the given selection query.

\[ \text{SELECT } * \text{ FROM } M \text{ WHERE Director = "Taika Waititi" AND (Year = 2000 OR Year > 2010)} \]

**ANSWER:** Hash index on Director. B+ tree index on Director alone, on Year alone, on (Director, Year), and more. Hash index on Year does not match due to the OR within that conjunct.

2. (3pts) Propose a physical query plan for this query that has a total I/O cost of \( 10^6 \) pages.

\[ \text{SELECT JoinDate, AVG(Age) FROM U GROUP BY JoinDate} \]

**ANSWER:** LQP is just \( \gamma_{\text{JoinDate}, \text{AVG(Age)}}(U) \).

PQP is a sort-based group by aggregate. One pass over \( U \) suffices, since all of \( U \) fits entirely in memory! So, we only need 1 pass over \( U \), with I/O cost of \( N_U = 10^6 \). Note that a PQP of hash-based group by aggregate also has the same cost, since the hash table with the running counts and sums fits in memory; its size is \((N_U \times 8/64) \times 3 \times 8KB \times 1.4 \approx 4.2GB\).

3. (5pts) Propose a physical query plan for this query that is fully pipelined and has a total I/O cost of \( 101 \times 10^5 \) pages.

\[ \text{SELECT Director, AVG(Stars) FROM R, M WHERE R.MID = M.MID AND Year > 2010 GROUP BY Director} \]
**ANSWER:** LQP is $\gamma_{\text{Director,AVG(Stars)}}(\sigma_{\text{Year} > 2010}(M) \bowtie R)$.

PQP has a filescan for both $M$ and $R$, followed by a hash join and finally, a hashing-based group by aggregate. The selection on $M$ is done on the fly during the filescan, and a hash table is built on the selected subset of $M$. The whole plan is clearly pipelined.

Since $M$ is much smaller than memory, the hash table easily fits in memory. Thus, we only need one pass each over $R$ and $M$, which leads to a total I/O cost of $101 \times 10^5$ pages.

4. (6pts) Propose a physical query plan for this query that has a total I/O cost of $111 \times 10^5$ pages.

SELECT AVG(Stars) FROM R, M, U
WHERE R.MID = M.MID AND R.UID = U.UID
AND Age <= 35 AND MName = "Avengers: Infinity War"

**ANSWER:** LQP has a right-deep join tree with the selections on $U$ and $M$ pushed through the joins and a projection on $U$ inserted before its join to reduce hash table size: $\gamma_{\text{AVG(Stars)}}(\pi_{\text{UID}}(\sigma_{\text{Age} \leq 35}(U))) \bowtie (\sigma_{\text{MName} = \"Avengers: Infinity War\"}(M) \bowtie R)$.

PQP has a filescan for all tables, with the selections on $U$ and $M$ done on the fly during the filescan, followed by two hash joins. For the first hash join, the hash table is built on the selected subset of $M$, which will likely have just one tuple, and thus, easily fit in memory.

For the second hash join, the hash table is built on the projected selected subset of $U$. Since this hash table only needs to store $\text{UID}$ values, its size is less than $(N_U \times 8/64) \times 8KB \times 1.4 \approx 1.4GB$, which also easily fits in memory. Thus, the whole PQP is fully pipelined and we only need one pass each over $U$, $M$, and $R$, which yields a total I/O cost of $111 \times 10^5$ pages.

**Q 5. [18pts] Query optimization.** We are given a relational database schema with the following three relations, wherein $A$ and $B$ are discrete (string) attributes, while $X$ is a numeric attribute.

$R(A,B,X)$, $S(B,C)$, and $T(A,C)$

For each of the following questions (3pts each), clearly circle either Yes or No for whether the given queries $Q_1$ and $Q_2$ are equivalent.

1. $Q_1 : R \bowtie (S \bowtie T)$ and $Q_2 : (R \bowtie S) \bowtie (S \bowtie T)$

   **Yes.** Joins are associative and self-natural join of $S$ in $Q_2$ is redundant.
2. $Q_1 : \sigma_{A=a, B=b, C=c}(R \bowtie S \bowtie T) \quad \text{and} \quad Q_2 : \sigma_{B=b}(R) \bowtie \sigma_{C=c}(S) \bowtie \sigma_{A=a, C=c}(T)$

**Yes.** Pushing the select partially to $R$ and $S$ is fine, since the natural joins implicitly enforce the other parts of the conjunctive predicate globally.

3. $Q_1 : \pi_{A,B}(R) \cup (\pi_A(T) \times \pi_B(S)) \quad \text{and} \quad Q_2 : \pi_{A,B}(R \bowtie S \bowtie T)$

**No.** The $\times$ in $Q_1$ could introduce new tuples not present in the full join.

4. $Q_1 : \sigma_{A=a, B=b}(R) \bowtie \sigma_{B=b, C=c}(S) \bowtie \sigma_{C=c, A\neq a}(T) \quad \text{and} \quad Q_2 : (T \cap T) - T$

**Yes.** $A = a \land A \neq a$ always leads to the empty set.

5. $Q_1 : \gamma_{A, AVG(X)}(R) \quad \text{and} \quad Q_2 : \gamma_{A, AVG(X)}(\pi_{A,X}(R))$

**No.** Deduplication on $(A, X)$ in $Q_2$ could reduce the counts for some groups.

6. $Q_1 : \gamma_{COUNT(DISTINCT B)}(R \bowtie S) \quad \text{and} \quad Q_2 : \gamma_{COUNT(DISTINCT B)}(R \bowtie T)$

**No.** The two joins might remove a different set of tuples from $R$.

**Extra Credit Q. [5pts]** Suppose you are given a Hadoop cluster. You are also given a set of tables in the Netflix schema discussed in class (also provided in Q 4 earlier). Each table is stored as a massive distributed CSV file on HDFS that is hash partitioned on its respective primary key. Assume all ID attributes are stored as integers and there are no majors skews in the distributions of all attributes.

You are asked to determine the total number of ratings for each movie in the Ratings table, as well as the total number of ratings for each user in the same table. But you should output only those movies and users that have at least 10 ratings each.

How will you implement this task in pure MapReduce with no SQL-oriented abstractions on top (Hive, SparkSQL, etc.)? A proper explanation of the Map and Reduce phases are enough; no need to write code in Java, Python, etc. If you had Hive or SparkSQL in the cluster, how will you implement this task using SQL?

**Answer:** In the Map function, emit pairs of IDs and counts as follows for each tuple in the Ratings table: $(UID, 1)$ and $(MID, 1)$. In the Reduce function, add up all the counts from the pairs received to obtain the total count per UID and per MID. Emit only those IDs whose corresponding counts are at least 10.

But the above implementation is buggy! Since UID and MID are both integers, the Reduce function can not distinguish between a UID value of, say, 42 from an MID value
of 42! The correct approach is to augment the IDs with a distinguishing indicator in the Map function. For instance, we can convert $UID$ into a pair $UID' = ('U', UID)$ and emit $(UID', 1)$; likewise for $MID' = ('M', MID)$. This ensures that the augmented pairs are distinguishable even if $UID$ and $MID$ are identical. These are the sort of subtle bugs caused by the lack of physical data independence! Alternatively, you could write two separate MapReduce jobs, with the first one focusing on $UID$ alone and the second, on $MID$ alone; in this case, no augmentation is needed because there is no ambiguity.

Here is the same task in two simple lines of SQL that works flawlessly regardless of how the data is encoded or laid out. Physical data independence FTW! :)

```
SELECT MID, COUNT(*) AS C FROM Ratings GROUP BY MID HAVING C >= 10
SELECT UID, COUNT(*) AS C FROM Ratings GROUP BY UID HAVING C >= 10
```